OST
Ostschweizer
Fachhochschule

Link Management Tool with Internet Archive Integration

Department of Computer Science
OST – University of Applied Sciences
Campus Rapperswil-Jona

Spring semester 2023

Authors	Benny Joe Villiger and Thomas Zahner
Advisor	Prof. Dr. Olaf Zimmermann
External Co-Examiner	Dr. Hans-Peter Hoidn
Internal Co-Examiner	Prof. Laurent Metzger

Abstract

Hyperlinks, also known as URLs or links, are the foundation of the Internet, they allow seamless navigation between online resources with a single click. They make the Web a web. However, the content behind a link can change, move or be deleted without notice. This, combined with the often short-lived nature of content on the Internet, can lead to problematic situations. The phenomena of disappearing content and unannounced changes are known as link rot and content drift respectively.

This thesis addresses the challenges of both link rot and content drift. We develop a command-line tool called Link Management Tool (LMT) to provide an automated approach to detecting and fixing broken links and drifted content. Additionally, we extend an existing free and open source link-checking tool called lychee. In both implementations, we integrate the Wayback Machine, a digital web archive, to provide access to archived snapshots of websites from the past.

Contents

List of Figures

List of Tables

Management Summary

Context

In today's digital age, the World Wide Web has become the indispensable platform for accessing and sharing information. At the heart of the Web are hyperlinks, also known as URLs or simply links. They are the threads that connect webpages and allow users to traverse the landscape of online content.

However, online content tends to be short-lived, and changes often go unnoticed. The phenomena of disappearing content and unannounced changes are known as link rot and content drift. Rotten or dead links are URLs pointing to resources that show obvious signs of malfunction. Content drift can result in linked information that is misleading or that differs significantly from the original intent of the linker. They can undermine the accuracy and credibility of websites, degrade the user experience and damage the reputation of the website or organisation. Checking and fixing links manually on a regular basis is time-consuming and error-prone.

Approach

We develop a command-line tool called Link Management Tool (LMT) providing a semi-automatic approach to detecting and fixing rotten links and drifted content. Additionally, we extend lychee, a free and open-source link-checker. We integrate the Wayback Machine, a digital Web archive, into LMT and lychee to obtain access to archived snapshots of websites taken in the past. The project is executed using an agile methodology, emphasizing flexibility, collaboration, and iterative development. This allows for continuous feedback, and a fast adaptation to potentially changing requirements.

Results

LMT was released to the public on GitLab under the MIT licence. The metadata snapshot feature provides functionality to store a website's metadata (such as the title). When checking a website again, those stored metadata can be compared to the current ones, which enables the detection of content drift. The modular design of LMT allows integrating it into scripts, providing the flexibility to schedule the link checking process at strategic intervals such as before each publication or release. Scheduling the link checking supports the continuous monitoring of links.

In addition, lychee was extended with a new feature that offers recommendations for broken URLs by utilising snapshots from the Wayback Machine.

Early user feedback and testing confirmed the reliability of LMT to identify and resolve broken links.

Acknowledgements

First and foremost, we would like to thank our project supervisor, Olaf Zimmermann, for his invaluable support and guidance, throughout the thesis. His expertise and mentorship helped us a lot during the project.

We would also like to express our gratitude to Matthias Endler from the lychee project for his valuable support. Thanks to his cooperation, the helpful discussions and his quick answers to our questions, we were able to successfully contribute to lychee.

Additionally, we would like to thank Thomas Villiger for proofreading this thesis and providing valuable suggestions. His attention to detail has improved the clarity of our writing.

1 Introduction

Hyperlinks, also called URLs or simply links, are the fundamental concept of the internet. They allow users to easily navigate between different resources with a single click. They essentially are what makes the Web a web. While the hyperlinks are the reason the internet is so powerful compared to traditional media, they also introduce a new problem. Content on the internet is hosted by a third party who has control over the content. This means that hyperlinks point to content that can be changed, moved or deleted entirely without notice. Such changes can accumulate over time and might make the original information inaccessible.

> This often-irreversible decay of web content is commonly known as link rot. It comes with a similar problem of content drift, the often-unannounced changes–retractions, additions, replacement–to the content at a particular URL. (Zittrain, Bowers, and Stanton 2021)

A study from 2014 found that link rot is a significant issue, within legal journals and even within U.S. Supreme Court opinions. It showed that over 70% of the URLs within three selected and well-known legal journals and 50% of the URLs within U.S. Supreme Court opinions are experiencing link rot and content drift. (Lessig, Zittrain, and Albert 2014)

While link-rotting is a phenomenon that cannot be stopped there are ways to mitigate the effects on one's own content and websites. There are automatic link checking tools that can detect broken links. Also, there are internet archiving projects that aim to capture and preserve web content, so that the original version can be accessed even if the web page changes. This thesis explores different ideas on how to alleviate the problem of linkrot and attempts to provide a technical solution for it.

Namely, the objective is to create a link checking tool or use an existing one and integrate it with the Wayback Machine, a well-known internet archive, in such a way that broken or rotten links can be detected and fixed.

2 Background

2.1 Broken link

A uniform resource locator (URL) also known as hyperlink, web address or simply link, is a string used to identify a resource. (World Wide Web Consortium 2009) The World Wide Web Consortium (W3C) defines URL validity, which refers to whether a given URL is correctly formatted and follows the syntax rules.

A URL is considered to be “broken”, “dead” or “rotten” when the underlying webserver is unavailable or returns an erroneous HTTP return status code.

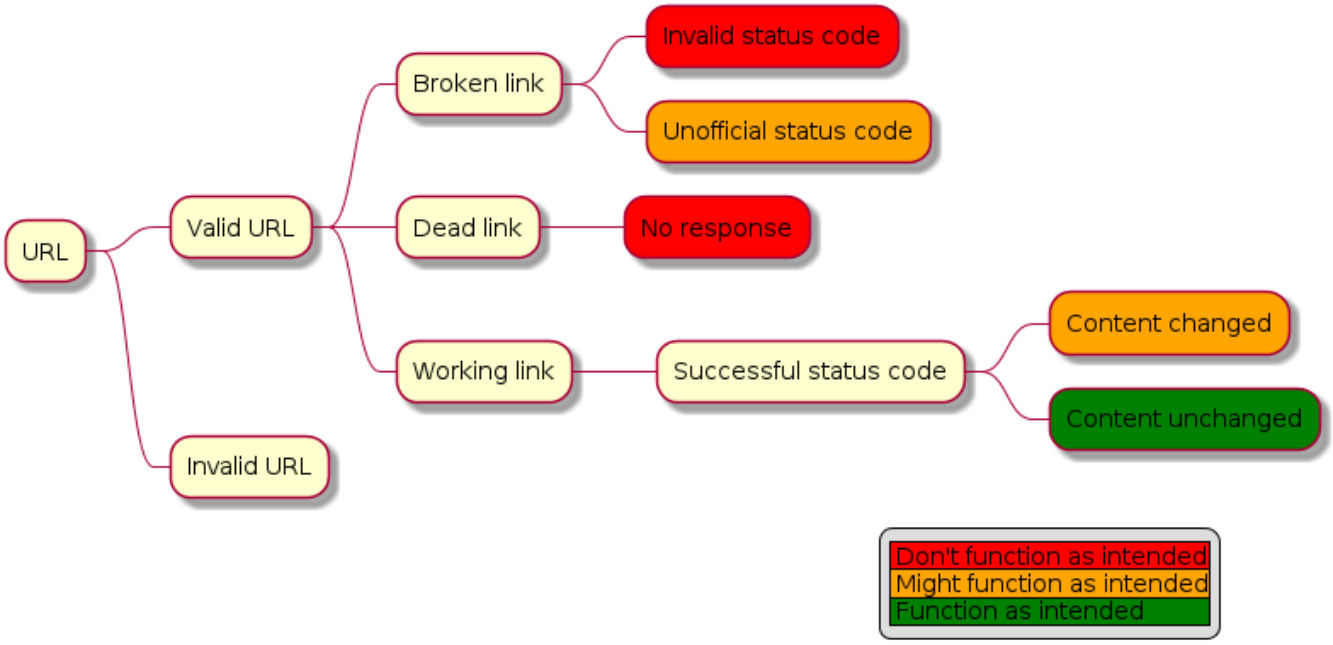

Figure 1: URL classification

Figure 1 illustrates the nomenclature used in this document in relation to links. This definition was taken into account when planning and implementing the tool, as described in the “Implementation” chapter.

At the top level a distinction between valid and invalid URLs, as defined by W3C, is made. Invalid URLs are not classified further as they are also not discussed further in this thesis. Any text or data that does not correspond to a valid URL can be considered an invalid URL. Valid URLs are split into three categories: broken links, dead links and working links. A link is considered broken if an invalid or unofficial status code is returned. When receiving an invalid status code, such as “400 Bad Request”, then the link most likely doesn’t work as intended. When receiving an unofficial status code, i.e. a code not defined in RFC 9110, it cannot be known if the link works as intended. If no response is received within a specified time frame, the URL is considered “dead”. Finally, if a successful status code is received, the link should work as intended. However, it is still possible that the content of the webpage has changed and no longer represents the original meaning of the article when it was linked. This phenomenon is known as “content drift”.

Table 1 contains examples for the link classifications used in Figure 1.

Table 1: URL classification

URL	Status Code	Type	Comment
I am not a link	-	Invalid URL	This text does not follow the syntax rules of W3C.
https://www.i-do-not-exist.ch	No response	Dead link	This URL is valid, but there's no server listening.
https://www.example.com/no-exist	404 (Not found)	Broken link	The page couldn't be found, but the server responded.
https://www.ost.ch/de	200 (OK)	Working link	

2.2 Content drift

A link is considered to be broken or rotten if there is any obvious sign of malfunction. However, those types of links are not the only concern.

They are accompanied by a more subtle but no less insidious phenomenon called "content drift". Content drift refers to unannounced modifications such as retractions, additions, or replacements made to the content associated with a specific URL. Such changes can make the content at the end of a URL misleading - or dramatically divergent from the original linker's intentions. (Zittrain, Bowers, and Stanton 2021)

2.3 Limitations

When checking Web links in practice, HTTP requests have to be made. When requesting a server with an HTTP request it can process the request in any way, and it can create any HTTP response, or even none at all. So while in theory HTTP status codes and their meaning are well-defined, in practice Web servers can return arbitrary status codes and their meaning can be misused. For example, a "400 Bad Request" or an unofficial status code might be returned even though the request was successful. Also, there are websites which tell the user that a page cannot be found, when navigating to a non-existent page, but still return a "200 OK" response instead of the designated "404 Not Found" response.

Additionally, many Web servers and websites today are stateful or require authentication. For example, a website may return a "401 Unauthorized" status code, when requesting a resource where authentication is expected but not provided. Some Web servers implement rate-limiting to prevent denial-of-service attacks which might lead to unpredictable return status codes, when checking multiple URLs of the same domain within a given time frame.

Because of all these reasons, the scheme depicted in Figure 1 is not applicable to every single URL and HTTP Web server. It should however be a good generalisation applicable to Web servers following the official meaning of status codes as defined in RFC 9110. Some limitations such as rate-limiting cannot be easily handled or only to some extent. These limitations must be taken into account in the planning and development of the tool.

2.4 Internet archives

A research paper from 2014 conducted a lifespan study of 10 million webpages that were collected in 2001. They found that more than 90% of the webpages had disappeared in the last 12 years and determined that the average lifespan of a webpage is just over three years, or 1,132.1 days. (Agata et al. 2014)

This study illustrates the severity of the link rot problem. To preserve content on the internet and to prevent the potential loss of knowledge and information websites can be archived. The most notable example of such a Web archiving project is the Internet Archive with their Wayback Machine.

The Internet Archive began archiving websites in 1996. As of 2016 they have stored over 510 billion time-stamped Web objects, also called Web captures. At that time they held 273 billion webpages from over 361 million websites, taking up 15 petabytes of storage. (Goel 2016)

With the Wayback Machine it is possible to easily access the Web captures of the Internet Archive. The Wayback Machine allows for the retrieval of any URL through a search bar, as depicted in Figure 2. If the specified URL has been captured in the past, a calendar view will display the available captures along with their respective dates and times. Users can then access these snapshots, as shown in Figure 3, enabling them to browse the website as it appeared during those specific points in time.

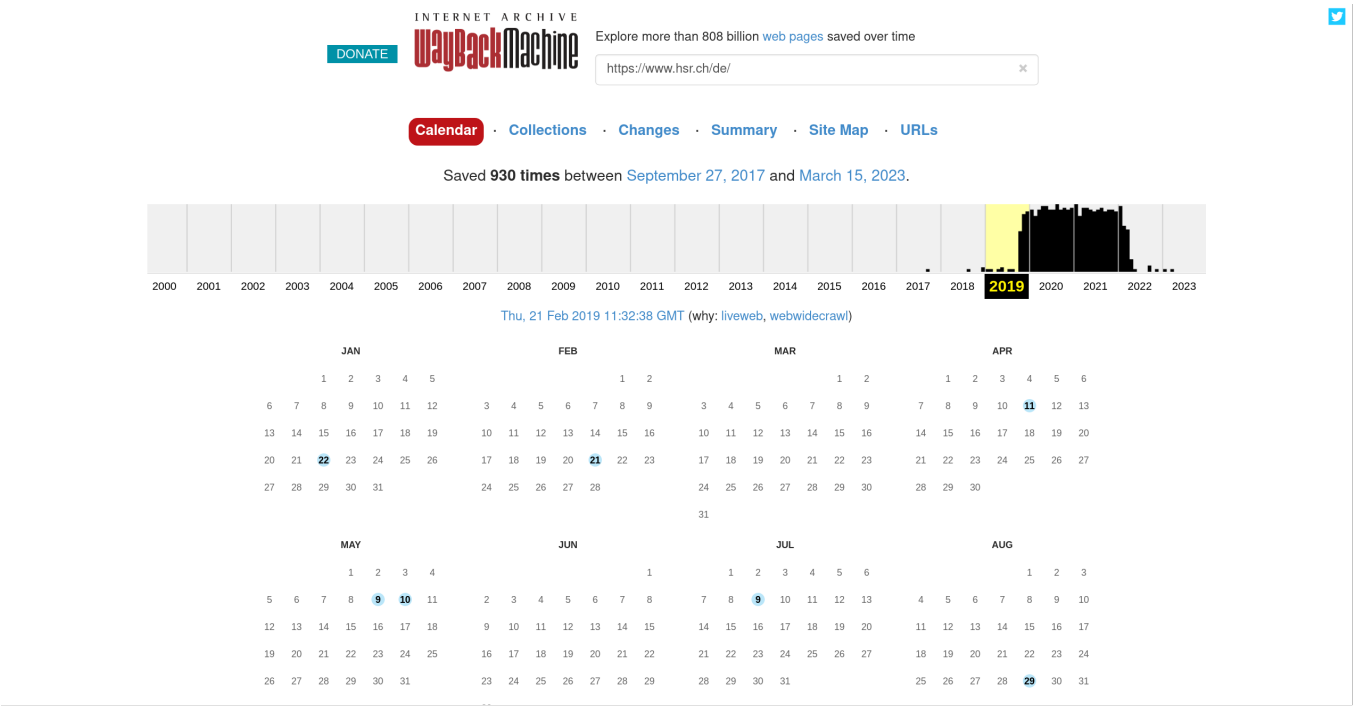

Figure 2: Searching for URLs with the Wayback Machine

The Wayback Machine also provides a publicly accessible RESTful HTTP API, allowing it to programmatically check if a given URL is archived and accessible in the Wayback Machine.

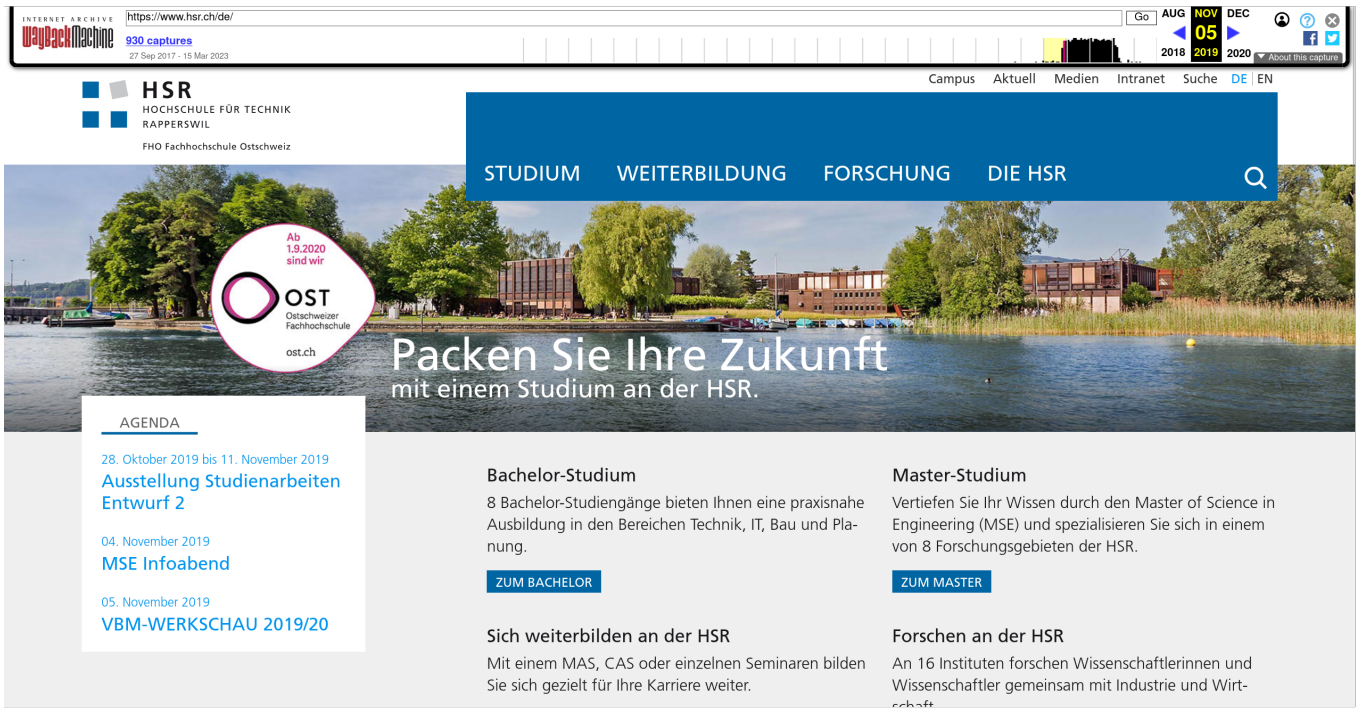

Figure 3: Browsing the Web of the past with the Wayback Machine

2.5 Introduction to lychee

lychee is a popular, free and open source link checking tool written in the Rust programming language[1]. It is able to find broken hyperlinks and email addresses in Markdown, HTML, reStructuredText and other text files and also supports checking websites. It focuses on being fast and resource-efficient by making use of asynchronous and stream-based mechanisms. lychee is available as a command-line utility, Docker image, Rust library and GitHub Action. [2]

In the course of this thesis, we will be providing further explanations as we extend the functionality of lychee.

[1] A general-purpose programming language that emphasises performance, type safety, and concurrency: https://rust-lang.org/
[2] Features of lychee: https://github.com/lycheeverse/lychee#features

3 Requirements

This chapter discusses the functional and non-functional requirements.

3.1 Functional requirements

For the functional requirements user stories are defined using two different personas.

Personas

Code owner

- A person who writes the source of their documents in a regular text file (e.g. Markdown or LaTeX).
- Uses a version control system to maintain the source files.
- Is technically proficient.
- Doesn't like to spend time on repetitive tasks and therefore automates everything as far as possible.

Content management system user (CMS user)

- Uses a content management system[3] (CMS) to write and maintain the content of their website or blog.
- Doesn't maintain source files locally.

User story 1

> As a code owner, I want to check the source code for *broken links* in an automated way, so that both the source code and the final artifacts (e.g. compiled program or built website) do not contain any *broken links* to ensure functionality and a good user experience while at the same time increasing productivity.

Importance: essential

User story 2

> As a code owner, I want to find and replace *broken links* with links that point to the intended content in an automated manner so that I don't have to manually check (and possibly replace) all the links in the repository.

Importance: essential

User story 3

> As a CMS user, I want to check my website for broken links so that visitors have an enjoyable experience.

Importance: essential

[3] Content management system, often abbreviated as CMS. WordPress for example is a well known CMS.

User story 4

> As a code owner or a CMS user, I would like to be notified of links where the underlying resource has changed in some way (e.g. owner or metadata changes), so that I can re-examine the resource and decide, if it is still an intended target.

Importance: optional

User story 5

> As a code owner I want to periodically check for broken links and receive suggestions for corrected links in the form of a merge request, all in an automated manner, so that the effort for link maintenance is minimised and the time saved can be spent on more important tasks.

Importance: optional

User story 6

> As a CMS user or a code owner working with LMT & lychee, I would like to see a list of all broken links that do not have a corresponding archived version, so that I can immediately start working on updating the links that require manual intervention.

Importance: optional

User story 7

> As a CMS user or a code owner, I want to be able to check links to websites even if they implement a rate-limiting mechanism, so that LMT & lychee work consistently and I don't have to manually check rate-limited websites.

Importance: optional

User story 8

> As a CMS user or a code owner, I want to be able to configure LMT & lychee with credentials to be able to check links to websites even if they require authentication, so that I don't have to check those websites manually.

Importance: optional

3.2 Non-functional requirements

This section describes the non-functional requirements and the approaches to meeting them.

Portability

The Link Management Tool (LMT) is usable on the current LTS[4] version of the following three major (x64 architec-

[4] Long-term support

ture[5]) desktop operating systems: GNU/Linux, macOS and Microsoft Windows.

Realisation

Making use of Deno and Deno's compile[6] command ensures that the aforementioned platforms are all supported.

Interoperability

The Link Management Tool (LMT) is capable of being integrated into development pipelines and is interoperable with other command line programs.

Realisation

The command-line interface (CLI) program achieves this interoperability by adhering to the GNU standard[7] for command line interfaces and by returning appropriate exit status codes[8].

LMT supports multiple common formats for input and output, such as JSON and CSV.

The output of the "fix" command is directly usable with the UNIX `sed`[9] command.

No reserved keywords are used as separator for formatted output (or input) of any command URL encoding.

Regulatory

The Link Management Tool (LMT) doesn't use unlicensed software components and required attributions are included in the documentation. Both LMT and lychee should prefer the use of software components licensed under free and open source licences. The licences should also be business friendly, meaning it should not force a company to publish their changes to the application. Namely, the MIT, Apache and Eclipse licences are fine while the GPL licence should be avoided.

Realisation

Dependency reports are generated for LMT. All licences of direct dependencies are checked and documented in this report.

Usability

LMT has clear and comprehensive documentation for each of the supported operating systems, including user guides, technical specifications, and developer documentation. Help commands with usage examples exist for each command. It doesn't take more than fifteen minutes for a technically proficient person to get used to the LMT.

Realisation

User testing is carried out to ensure that the above requirements are met.

[5] 64-bit version of different processors: https://en.wikipedia.org/wiki/X86-64
[6] Compiling executables with Deno: https://deno.land/manual@v1.31.1/tools/compiler
[7] GNU Standards for Command Line Interfaces: https://www.gnu.org/prep/standards/html_node/Command_002dLine-Interfaces.html
[8] Exit status codes: https://www.gnu.org/software/bash/manual/html_node/Exit-Status.html
[9] A UNIX command to perform functions on files, including searching and replacing. SED stands for "stream editor": https://www.gnu.org/software/sed/manual/sed.html

Maintainability

It is possible to detect and diagnose a bug of any kind in one working day.

Changes to the source code don't break existing features.

Realisation

SonarCloud[10] is used to provide a measure of the source code's quality. All measures of Sonar, that is reliability, maintainability and security, must be graded with the highest grade in the final product.

Additionally, automated unit- and integration testing is used to ensure the quality of the product. Deno's coverage[11] command is used to measure test coverage. Git[12] and GitLab[13] are used for source code version control and every change to the source code or documentation on the main branch is reviewed by one of the team members using merge requests.

Reliability

LMT provides accurate and reliable link-checking results, meaning links are classified correctly as described in the broken link definition, when used by the code owner or CMS user persona.

Realisation

Error handling, unit, integration and user tests ensure that the program works when used in an intended way. Being able to configure LMT, allows the user to work with LMT in different ways.

[10] Software for continuous code quality and security: https://www.sonarsource.com/products/sonarcloud/
[11] Collecting test coverage with Deno: https://deno.land/manual@v1.31.1/basics/testing/coverage
[12] Free and open source distributed version control system: https://git-scm.com/
[13] Free and open source DevOps software package: https://about.gitlab.com/

4 Architecture

4.1 Technical decisions

In this section the most important technical decisions taken at the beginning of the project are described.

Programming language

The decision of the programming language is summarised with the following Y-statement [14]:

> In the context of the Link Management Tool, we decided to use TypeScript as the programming language and against JavaScript, Kotlin and Rust to achieve the most risk-free developer experience because we are the most experienced with TypeScript, while at the same time guaranteeing a fully typed code base, accepting the downside of having to choose an additional runtime environment.

Runtime environment

Since it was previously decided to use the TypeScript programming language, a runtime environment had to be chosen. The following three major frameworks were considered:

- Deno [15]
- Node.js [16]
- Bun [17]

The decision for the best fitting technology is made using an evaluation table (Table 2) which consists of criteria with different weights. For each criterion, a score between 0 and 2 is set. The final score consists of all criteria scores, multiplied with their weight and summed up.

Table 2: Evaluation table technology

Topic	Weight	Deno	Node.js	Bun
Maturity	3	2	2	0
Maintainability	3	2	1	2
Third-party integration	2	2	2	2
Performance	2	2	0	2
Experience	1	0	2	0
Compile to executable	1	2	0	2
Score	Max=36	22	15	16

[14] A light template for architectural decision capturing: https://medium.com/olzzio/y-statements-10eb07b5a177
[15] V8-based runtime for JavaScript, TypeScript and WebAssembly written in Rust: https://deno.land/
[16] V8-based runtime for JavaScript written in C++: https://nodejs.org/
[17] JavaScriptCore-based runtime for JavaScript and TypeScript written in Zig: https://bun.sh/

Each criterion is explained in more detail below. The final decision can be summed up with the following Y-statement [18]:

> In the context of the TypeScript runtime environment for LMT, facing the need to easily create a compiled and fast executable, we decided to use Deno and neglected Node.js and Bun, to minimise the time and effort required for set-up and maintenance, accepting a small risk of encountering problems due to lack of experience.

Maturity While Node.js and Deno can be considered mature and stable as of 2023, Bun is still not quite complete yet. The README of the official repository states: "Bun is still under development."

Maintainability The setup for both Deno and Bun are easier compared to Node.js. Both support compilation and execution of TypeScript without additional configuration. With Node.js additional manual configuration is required.

Third-party integration All three of the frameworks are compatible with the huge NPM registry[19] therefore third-party integration should not be a problem.

Performance For performance comparison, the denosaurs/bench repository was consulted. This benchmark is automatically updated on a daily basis with the latest framework versions. In this benchmark Deno performs the fastest (100%) closely followed by Bun (92%) and Node.js (54%).

Experience Both team members have worked with Node.js before but not with Deno or Bun. Since the different runtime environments only affect how the project is configured, for example with a package.json file, but not how the program code will look (with a few exceptions) the experience was given less weight.

Compile to executable Node.js doesn't have a built-in feature to compile a project into a binary executable. Deno and Bun both support compilation into a self-contained executable without requiring any third-party programs.

Contribution to lychee

In the third week of work on the bachelor thesis, we searched for existing link checking tools. Most of the tools we came across were either proprietary or only available as web applications. There are however quite a few free and open source tools available. One of the most notable projects is the lychee project. Its README file also contains an informative feature comparison table of different link checking tools. This comparison table makes clear how powerful and configurable lychee is.

[18] A light template for architectural decision capturing: https://medium.com/olzzio/y-statements-10eb07b5a177
[19] Default package manager of Node.js: https://www.npmjs.com/

As the task assignment allowed us to extend or integrate existing link checking tools we decided to do this with lychee. Firstly, we made the Link Management Tool usable in combination with lychee. Secondly, we contributed to lychee itself to fix problems and extend its functionality. More information can be found in the "Implementation" chapter.

There were multiple reasons why we chose to extend and integrate lychee:

- lychee is a fast and feature-rich link checking tool
- it is already well-known and established
- it has an active maintainer and community
- it is licensed under the MIT and Apache licences
- by integrating lychee, its functionality and configurability can be leveraged

4.2 C4 Architecture model

The following diagrams from the C4 Architecture model are used to visualise the structure and relationships of LMT at different levels of abstraction. We have chosen to exclude the class diagram from the C4 Architecture Model in this document, as the level of detail provided in the component diagram is sufficient for the purpose of this document.

System Context

An overview of LMT, its users and the systems it interacts with is given by Figure 4. The personas “Code Owner” and “CMS User” are described in the persona section of the functional requirements chapter.

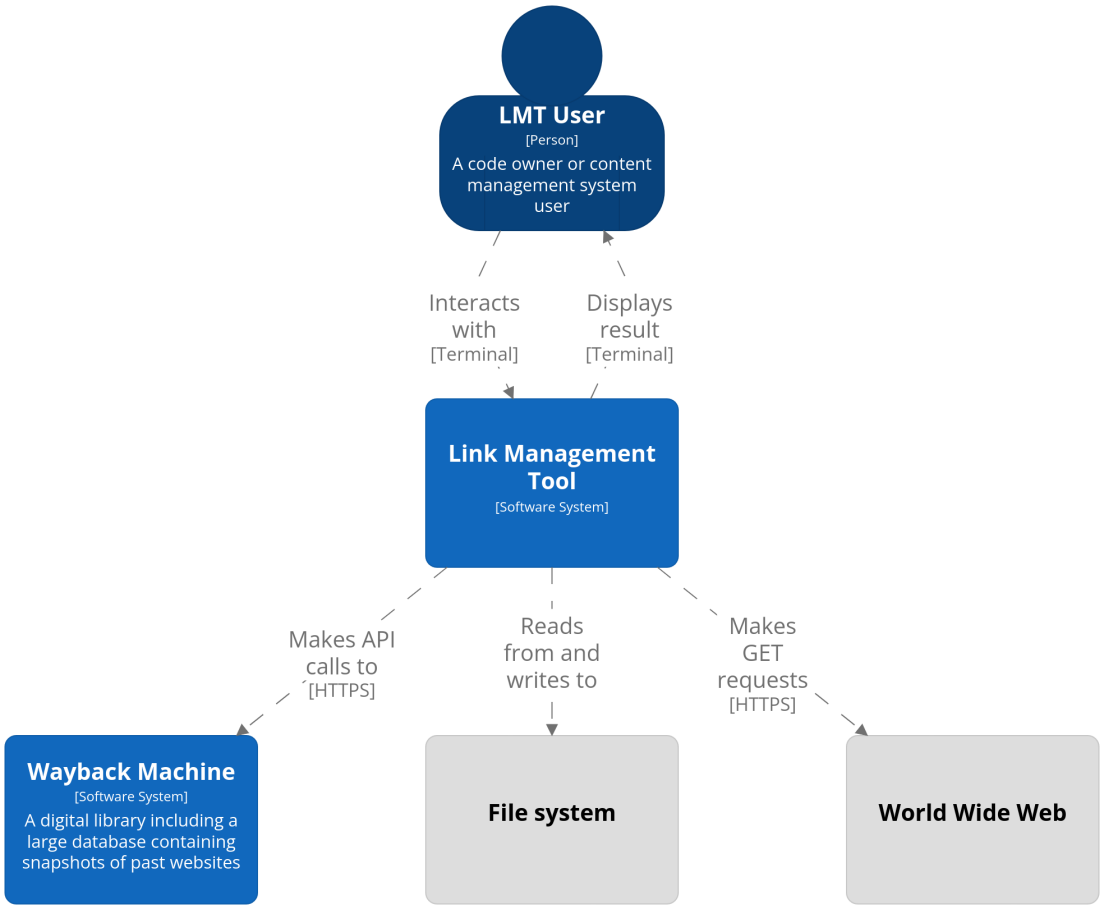

Figure 4: System Context Diagram

Container

The container view depicts a closer look into the system. Each container represents a separate application, which in this case includes lychee and LMT. Figure 5 especially shows the relationship between lychee and LMT.

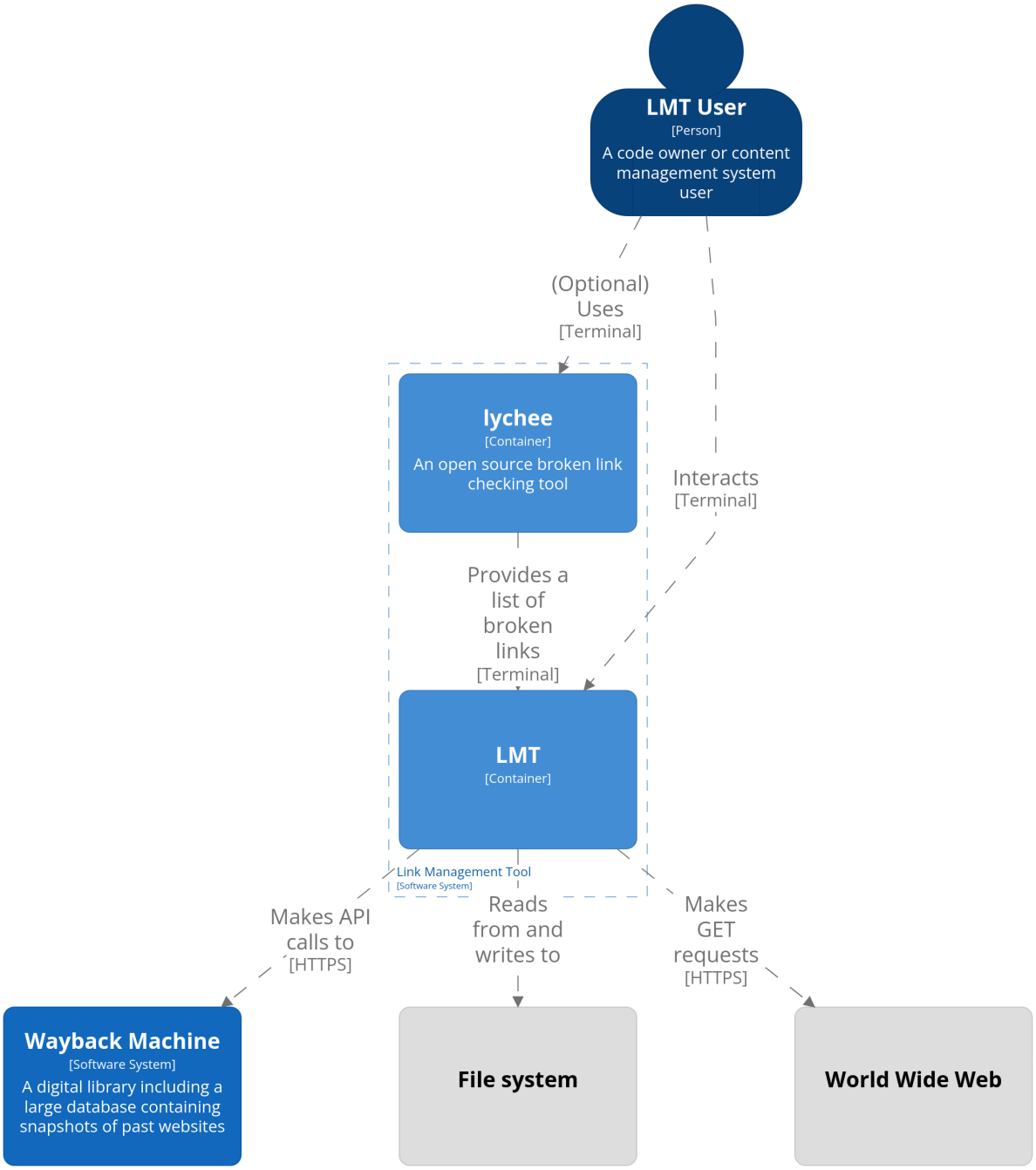

Figure 5: Container Diagram

Component

Figure 6 depicts a closer look into the internal structure of the LMT container.

Every interaction between a user and LMT begins in the “Entrypoint” component. The “Entrypoint” is responsible to accept a command through the command-line, validate it using the “Input Error Validation”, and invoke the corresponding “Command”. The “Fix Command” component depends on “Suggest Command” and “Check Command”, it combines the functionalities of both and is capable of replacing links in files on the “File System” using the “File System Facade” component, which abstracts common file operations. The “Suggest Command” component provides functionality to suggest link replacements available in the Wayback Machine. It checks the availability of links using the “Wayback Machine Facade” which uses the “Fetch Service”, abstracting calls to the “Wayback Machine” API. The “Check Command” component provides functionality to check files and websites for broken links, which implies the reading of files from the file system, using “File System Facade” and fetching web content, using “Fetch Service”. The “Suggest Command” and “Check Command” components both rely on the “Config Component”, which is responsible for reading configuration files using the “File System Facade”. Finally, the “Snapshot Component” provides functionality for creating and reading snapshot files, used by the “Check Command” Component”.

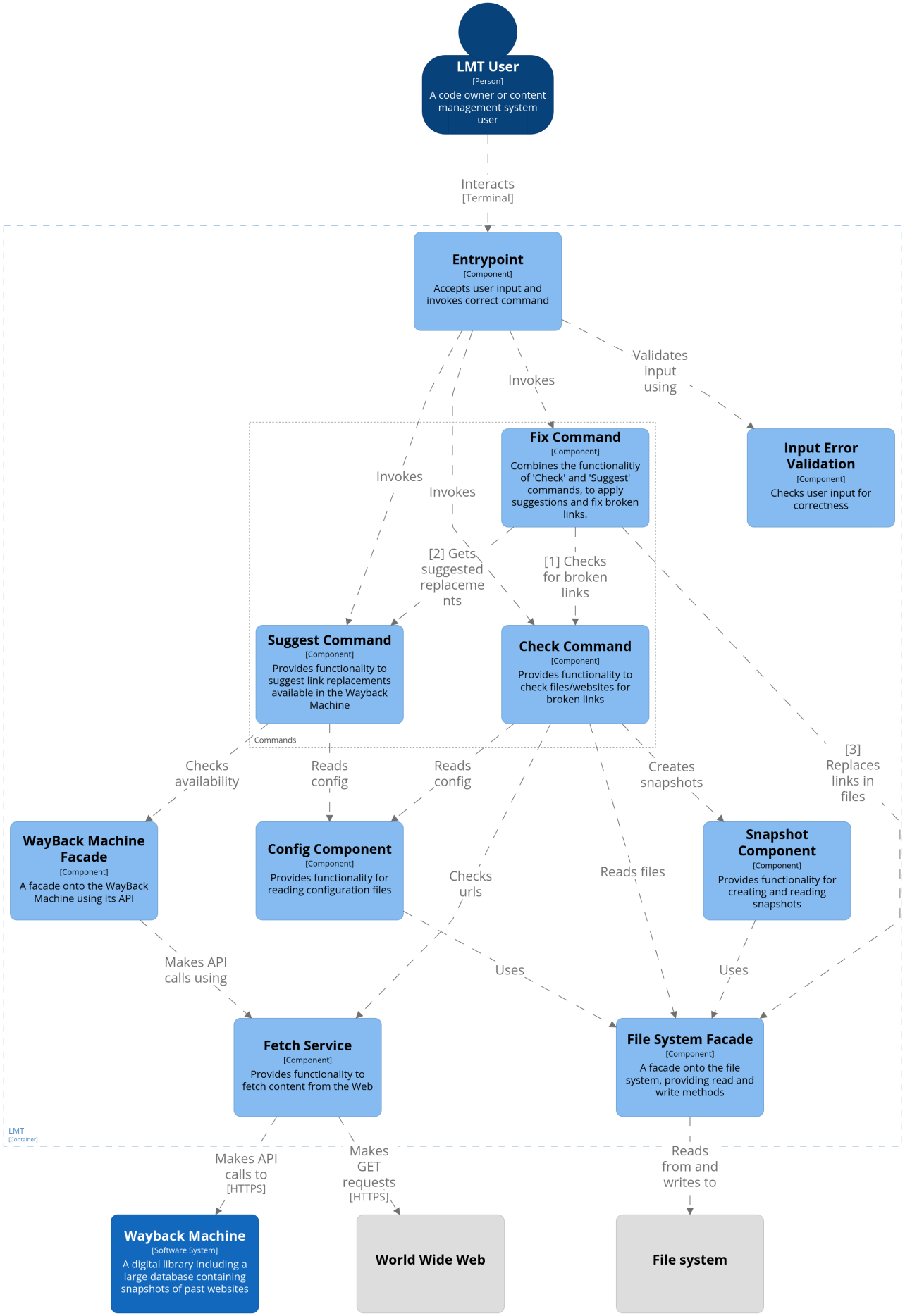

Figure 6: Component Diagram

5 Implementation

A big part of this project is the implementation and design of the Link Management Tool (LMT) and the contributions to the lychee project. This chapter describes the implementation details of LMT and the additions to lychee.

We decided to split the functionality of LMT into three commands: `lmt check`, `lmt suggest` and `lmt fix`. Each command can be used alone or in combination with the other commands. By using three different commands the different concerns are separated. By allowing the configuration of different input and output formats, the commands can be combined and used in a flexible manner. For example, it is possible to chain all three commands, and it is possible to use the suggest and fix command in combination with lychee.

5.1 Check command

Figure 7 shows the most important steps of the “check” command.

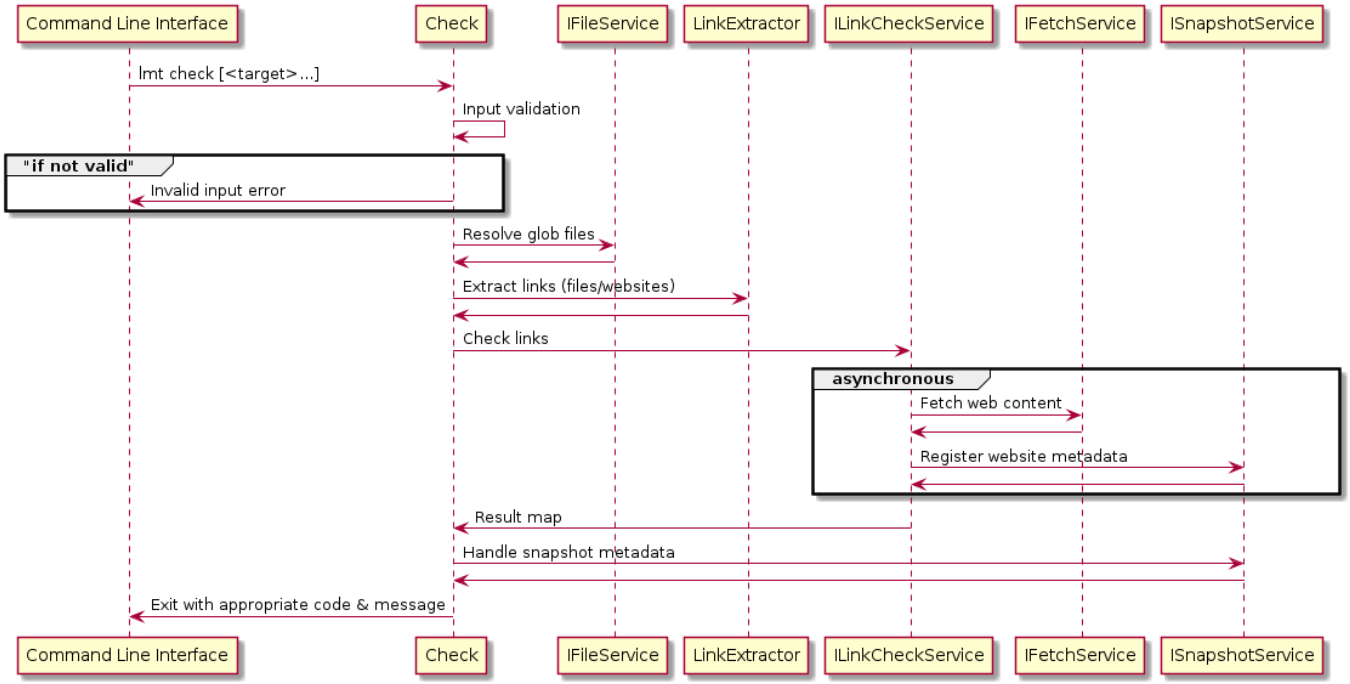

Figure 7: Sequence diagram of the check command

When the user executes the command `lmt check [<target>...]` the functionality of the “Check” class is invoked. At the very first, the user input is validated. If the input is valid, the Check class first resolves the files and glob file patterns. If a file does not exist, or the user input is not valid, LMT will exit early with exit code 1 and the corresponding error message. Next, the “Check” class invokes the “LinkExtractor” which gathers the links from the provided targets. The LinkExtractor is selected depending on whether the web flag has been provided (`--web`). If the flag is set, then the URLs are queried and extracted with the “HtmlLinkExtractor”. Otherwise, the URLs are extracted from the local files with another class, implementing the “LinkExtractor” interface, which is selected depending on the file extension. For example, a Markdown file (.md file extension) is extracted by the “MarkdownLinkExtractor” class, while a HTML file (.html) is extracted with the “HtmlLinkExtractor” class. Both of these use a third-party parser to first parse the files into a tree of nodes, where links can easily be searched for. For any other file extension, a fallback “PrimitiveLinkExtractor” is used, which uses a regular expression to find the URLs in files.

Once all the links have been extracted from the targets (files or web pages), the links are checked using “ILinkCheckService”. “ILinkCheckService” creates HTTP requests with the help of “IFetchService”. After the content of a page has been fetched, metadata are extracted with “ISnapshotService”. Both steps are performed asynchronously, in a non-serial and non-blocking manner, so that many websites can be scanned simultaneously. When all websites have been checked, a result map is returned from “ILinkCheckService”. The “Check” class then handles the previously registered metadata, if either the `create-snapshot` or the `verify-snapshot` flag was provided.

Finally, the check command returns an appropriate exit code and message, depending on the value of the result map and the handling of snapshot metadata. The output format can be configured with the `--output-format` flag.

LMT supports two different output formats for the check command, the “default” output format is a simple string containing newline-separated URLs, and is therefore easier for humans to read, while the “json” output format is more detailed and easier to be reused by other programs. The input- and output formats are explained in more detail later in this chapter.

Result map

The result map is the data structure used to represent the result of the check command. It is used as an internal representation of LMT but can also be shown when specifying the output format as JSON with: `--output-format json` The structure is inherited from the lychee project and was reused for LMT. On the topmost level the result map contains a `fail_map` and a `success_map`, where all broken links are contained in the `fail_map` and functional links are contained in the `success_map`. The results in both maps are grouped by their target name, i.e. the corresponding file name or URL.

The following example of a result map is obtained by running `lmt check --web https://example.com --output-format json`.

```
{
  "fail_map": {},
  "success_map": {
    "https://example.com": [
      {
        "url": "https://www.iana.org/domains/example",
        "status": 200
      }
    ]
  }
}
```

In summary, https://example.com contains a single link. This link returns a 200 OK status code, meaning the link is not broken.

Supported file formats

Supported and tested file formats for the check command are: Markdown, HTML and LaTeX. As shown previously, the link extractor is responsible for extracting URLs from files. There are dedicated link extractors for HTML and Markdown files. The appropriate link extractor is selected depending on the file extension. To extract links from HTML or Markdown files, the files are parsed with a parser library.

HTML and Markdown are both structured languages with syntax and rules, so URLs can for example be embedded within nested elements. To capture this complex structure and the potential URLs within those structures a parser is required.

Formally speaking, HTML and Markdown are not a regular language so they cannot be detected by a regular expression. Therefore, it is necessary to use a parser that is capable of recognising additional grammar structures. This concept of different levels of grammar classes is formally defined by the Chomsky hierarchy.

The following example illustrates, an example link within Markdown.

```
[link description](https://example.com/home).
```

Given the syntax of Markdown, the above example should point to the URL “https://example.com/home”. However, when ignoring the Markdown syntax and using a regular expression to detect a URL, there is no way of knowing whether the trailing “).” characters belong to the URL or if they have a Markdown specific meaning.

When the check command is used with unsupported file formats, a “primitive” link extractor is used as a fallback. This link extractor uses a regular expression to try to recognise URLs and works well for some file types, but can also produce inaccurate results. For example, recognition of LaTeX files proved to be accurate.

Status code handling

The check command considers all HTTP status codes in the range between 200 and 299 to be “working”. This is because Deno’s `Response.ok` property from the `fetch` result is reused internally. The living standard for fetch is defined by Web Hypertext Application Technology Working Group (WHATWG). This definition of “working” URLs is used, because it is suitable for most use cases.

A configuration file can be used if a user wishes to override this behaviour for specific URLs and status codes. More information can be found in the config file section.

Metadata verification

To realise user story 4 we created a “metadata snapshot” function. Two additional command flags are added to the check command:

```
-c, --create-snapshot  create a snapshot and write a file with the specified
    name. Default value: snapshot.json
-v, --verify-snapshot  verify that the specified snapshot matches the current
    metadata. Default value: snapshot.json
```

A snapshot refers to a file which records certain metadata for each URL checked with the `check` command. Snapshot files are written in the JSON format. Each snapshot file contains a “version”, “name” and “snapshot” property. The “version” and “name” properties are used to determine the metadata extraction method. At the moment, the only recognised combination is the name “website-title” with version number 1. The two fields ensure extensibility and future backward compatibility. The “snapshot” property is an object containing the metadata extracted from each URL. Below is an example of a snapshot file where two URLs inside a file have been checked. The following command was used to create the snapshot file: `lmt check MyFile -c`

```
{
  "version": 1,
  "name": "website-title",
  "snapshot": {
    "https://about.gitlab.com/": "The DevSecOps Platform | GitLab",
    "https://api-patterns.org/": "Microservice API Patterns"
  }
}
```

The “website-title” extraction method extracts the value of the HTML title tag[20]. HTML title tags are used by webpages to display the title bar in the browser.

[20] HTML title tag: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/title

If the title of a website changes, it can be assumed that the content may no longer reflect the original intentions of the linker. Of course, this assumption is not perfect, and there may be changes to the content of a webpage without a change to the title, and vice versa. However, testing has shown that the title tag is a fairly reliable indicator of content drift. In addition, the functionality is designed to be extensible so that different metadata extraction methods can be easily implemented in the future.

To make use of a snapshot file, the "check" command can be rerun with the `-v` flag to verify that the metadata of the webpages do not differ from the metadata stored in the snapshot file. If the metadata of the websites remain unchanged the command will exit successfully. However, if a change is detected, LMT will exit with an exit code value of 3, as defined in exit codes. When `lmt check MyFile -v` is run and one of the website titles has been changed, LMT's output may look as follows.

```
1 The following snapshot values didn't match the actual values.
2
3 [MyFile]: https://about.gitlab.com/
4 Snapshot value: The DevSecOps Platform | GitLab
5 Actual value: A totally new website
```

Output formats

The `--output-format` (`-o`) flag specifies how the output of the check command is formatted. Both formats can be reused by the suggest command:

- Default
- JSON

5.2 Suggest command

Figure 8 illustrates the steps involved when running the suggest command, starting with the input of the suggest command in the command line interface (CLI) and ending with the output to the CLI.

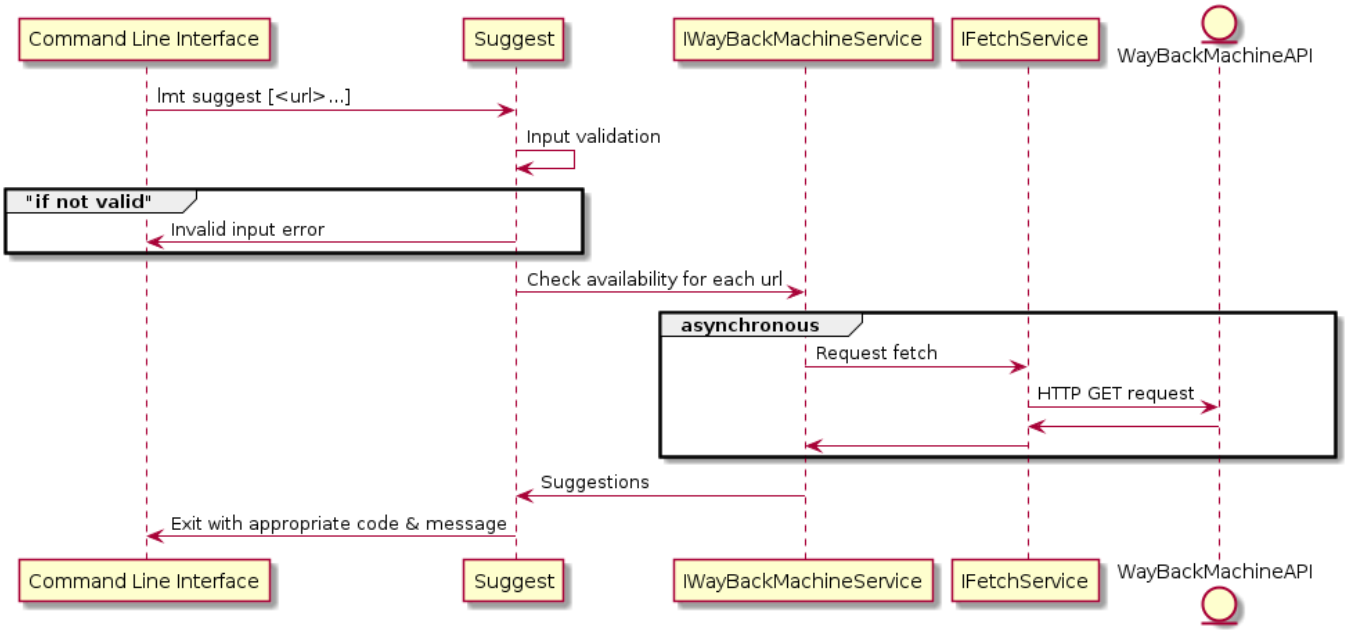

Figure 8: Sequence diagram of the suggest command

When the suggest command is invoked, one or more URLs are expected as input. The input must be in one of the input formats described later in this chapter. An input format can be changed using the `input-format` flag. At the beginning, the input is validated. An invalid input leads to the program exiting with an exit code of 1. Per default, the URLs are expected to be space-separated, for example `lmt suggest link1 link2`. After this, the URLs are extracted from the input, which then are used in the further process. The "IWayBackMachineService" interface expects an inheriting class to implement the `checkAvailability` method, which takes a URL as argument and requests a response from the Wayback Machine API[21] using "IFetchService" for the request, which includes whether the Wayback Machine has stored a snapshot of the URL in question, and if so, it includes a timestamp and the URL to the snapshot.

A GET request to http://archive.org/wayback/available?url=example.com, will return the following:

```
{
  "archived_snapshots": {
    "closest": {
      "available": true,
      "url": "http://web.archive.org/web/20130919044612/http://example.com/",
      "timestamp": "20130919044612",
      "status": "200"
    }
  }
}
```

[21]Wayback Machine API: https://archive.org/help/wayback_api.php

If the URL has been archived in the past by the Wayback Machine, then the URL pointing to the snapshot is added to the `suggestLinkMap`. The `suggestLinkMap` is a list of key-value pairs, where the key is the checked URL and the value is the response of the API.

Finally, when all URLs were checked, the suggestions are formatted according to the output format, which can be defined using the `output-format` flag, and written to the standard output.

Input formats

The `--input-format` (`-i`) flag specifies the format of the input which is expected by LMT. For example by adding `--input-format json`

The suggest command supports the output of the check command, which means that the input formats are the same as the output formats of the check command:

- Default
- JSON

Output formats

The `--output-format` (`-o`) flag specifies how the output of the suggest command is formatted. The flag expects one of the following values:

- Default
- JSON
- CSV
- Sed

5.3 Fix command

The `fix` command combines the check and suggest commands to automatically replace all the broken links with the received suggestions in all the provided files. It takes a list of files or glob patterns as input and then runs the replacement procedure.

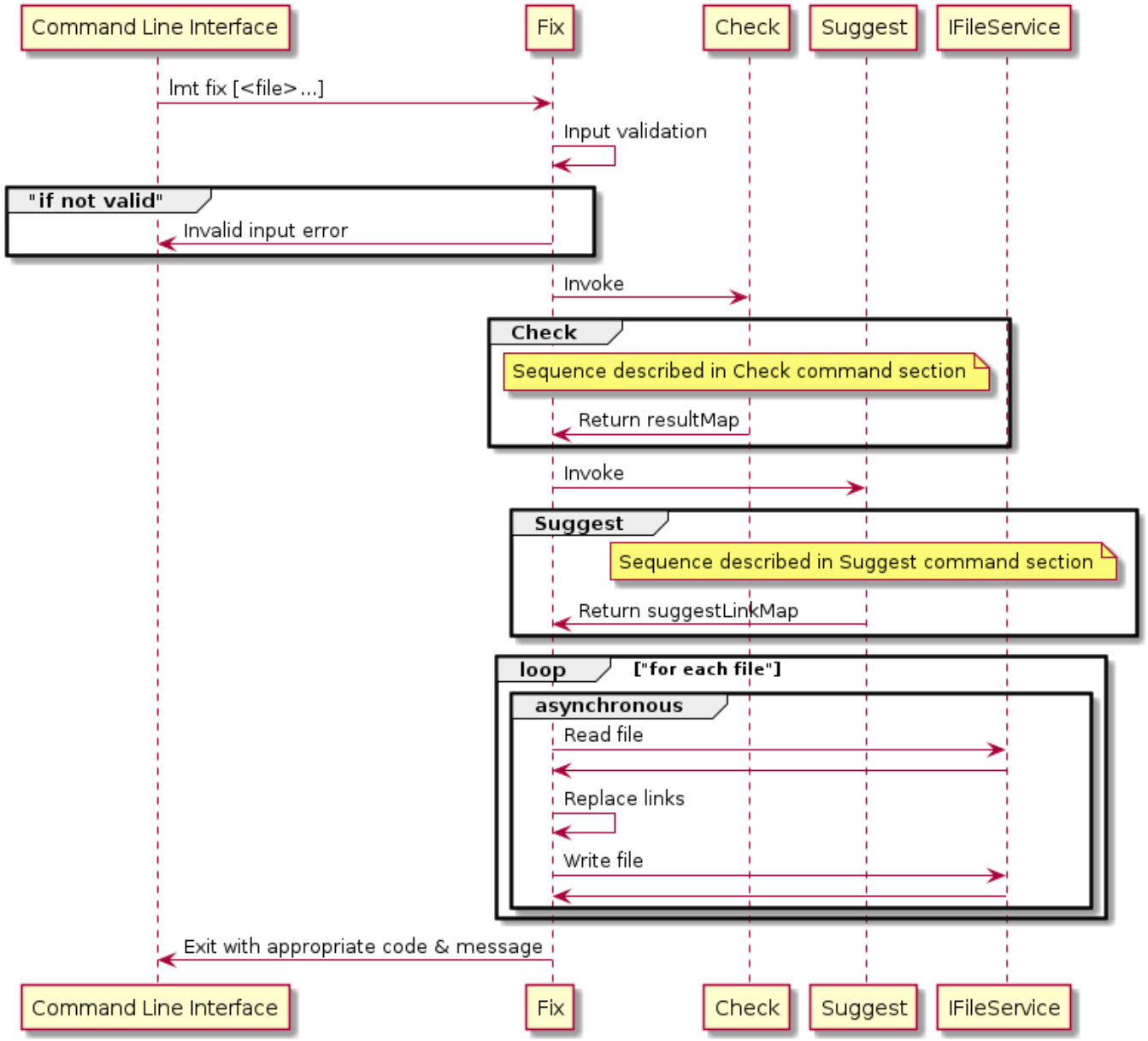

Figure 9: Sequence diagram of the fix command

When the fix command is invoked, first the input is checked for its correctness by the “Fix” class, which will return an Error when the input is not valid. “Fix” expects one or more files or glob patterns.

If the input is valid, “Fix” invokes “Check”, which checks the entered files for broken links. If the ResultMap returned by “Check” includes broken links, “Suggest” will be invoked, providing the `fail_map` as an argument.

Finally, if the `suggestLinkMap` returned by “Suggest” is not empty, and therefore contains replacement suggestions, “Fix” will read the affected files, using “IFileService” and replace the broken link occurrences with the corresponding suggestions.

Link map

It is possible to provide a link-map with the `--link-map` or short `-l` flag. The link map is a mapping of broken- to replacement-links.

If a link map is defined, only links in this mapping will be replaced in the given files, for example:

`lmt fix -l "https://www.example.com https://www.ost.ch"README.md` replaces all occurrences of the URL https://www.example.com with https://www.ost.ch in the `README.md` file. Multiple mappings are expected to be newline separated, such as:

```
1 lmt fix -l "https://www.example.com https://www.ost.ch
2 https://www.google.com https://www.bing.com" readme.md
```

5.4 Configuration file

LMT recognises `.lmt.json` as a configuration file if it is located in the same directory where LMT is run. This path is currently not configurable. The configuration file must be valid JSON. The root object must contain the `"ignore"` and `"ignoreMetadataForUrls"` property.

With the `"ignore"` option it's possible to specify status codes which should be ignored for a given URL. It's also possible to ignore all status codes with the option `"all"`. The value of `"ignore"` is an object, where all keys specify a URL as `string` and all values are of type `StatusCodes`. `StatusCodes` is either an array of status codes or `"all"`. (`type StatusCodes = Array<number> | 'all'`)

The `"ignoreMetadataForUrls"` option specifies an array of URLs which should be ignored when using the check command with the `--create-snapshot` or `--verify-snapshot` flag. This can be useful, for example, when using a permalink to the latest version of a product where new releases are expected at regular intervals.

In the following example, three URLs are configured to be ignored by LMT. For the last URL all status codes should be ignored, including DNS errors, connection errors, etc. so that effectively the whole URL is ignored. Additionally, the metadata snapshot functionality should be disabled for one URL.

```
{
  "ignore": {
    "https://example.com": [
      418,
      500
    ],
    "https://example.com/something": [
      404
    ],
    "https://deno.land/std@0.182.0/": "all"
  },
  "ignoreMetadataForUrls": [
    "https://gitlab.com/lmt7360522/lmt/-/releases/permalink/latest/"
  ]
}
```

5.5 Exit codes

To facilitate the use of LMT with automated scripts and integration with CI/CD pipelines, two special exit code values were defined. Exit codes indicate the state of the program after completion and allow pipeline automation and scripts to react accordingly. Table 3 depicts every possible exit code of LMT.

Table 3: Exit codes

Exit Code	Meaning
0	Success
1	Unexpected error or user error
2	Broken link(s) detected (`lmt check`)
3	Snapshot mismatch detected (`lmt check`)

5.6 Integration with CI/CD Pipelines

LMT is designed to integrate seamlessly with Continuous Integration and Continuous Deployment (CI/CD) pipelines. Using the "check" command for example, ensures that broken links are caught early in the pipeline, preventing them from reaching production environments. Any command of LMT can be easily integrated into different stages of a pipeline.

Docker image We have published a Docker image containing the binary executable of the Link Management Tool on DockerHub, which allows LMT to be easily integrated into CI/CD pipelines.

General script When using the Docker image, any LMT command can be run. For example `lmt check < filename(s)>` can be used to check for broken links.

Failures If a use case requires the pipeline to use or return the correct exit code, `set +e` should be added at the beginning of the script, so that it will not exit immediately, if a non-zero exit code was returned by a command. This could be the case, if a script should behave differently, when broken links have been detected or when a pipeline is allowed to fail on occurrence of specific exit codes.

The user guide contains sample files for GitLab pipelines and GitHub actions.

5.7 Dependencies

To provide transparency and facilitate licence compliance, we examine the direct dependencies of LMT and its associated licences. Table 4 lists all the direct dependencies of LMT.

Table 4: Dependencies

Name	Usage	Version	License
Alosaur Deno Land	Dependency Injection	0.38.0	MIT
Deno std Deno Land	Deno Standard Library	0.182.0	MIT
Deno dom Deno Land	HTML parser	0.1.38	MIT
Rusty markdown Deno Land	Markdown parser	0.2.2	MIT
Progress Deno Land	Progress bar	1.3.8	MIT

5.8 Contribution to lychee

This section documents all the changes, additions and bug fixes that have been made to the lychee project.

Since the additions to lychee were not a specific requirement for this bachelor thesis, and the benefits of using lychee were only discovered during the thesis, a first contact had to be made before considering contributing to this open source project. The first contact took the form of a discussion on GitHub, followed by a video call between us and the maintainer of lychee.

In the end four pull requests were made. They were all accepted and merged into the master branch. The changes to lychee are listed and explained below.

- Use standard error for error output
- Wayback Machine integration
- Concurrent archives
- Status codes in maps

Use standard error for error output

While trying to make LMT compatible with lychee, we ran into a problem. When piping the standard output of lychee into other programs, we noticed that lychee printed information to the standard output it shouldn't. This included the progress bar and various informative messages for the user. In practice, this meant that LMT would receive invalid JSON with the following command.

```
1 lychee deno.json -f json | lmt suggest -i json
```

As a quick first workaround, all characters before the first valid JSON character were filtered out. Additionally, we filed a bug report. After researching, we found that POSIX defines *standard error for writing diagnostic output*. This means that standard error is, unlike its name suggests, not only intended for writing errors but also for writing diagnostic information, like progress messages.

After a brief discussion with the lychee community, it was agreed that printing the affected information to standard error instead of standard output was the optimal solution. The changes were kindly made by lychee's maintainer, after which we were able to test the changes. After successful testing, it was possible to remove the workaround in LMT.

The changes were merged to lychee's master branch on April 11th: Use standard error for error output. The changes make it possible to pipe lychee's output into LMT or other programs and scripts, without the need for a workaround.

Wayback Machine integration

Creating an integration for the Wayback Machine in lychee was our main contribution. Before our contribution it was possible to detect broken links as follows.

```
echo 'https://google.com/jobs.html' | lychee -
```

This command behaves as if lychee was checking a file containing the single link "https://google.com/jobs.html". The provided link is no longer valid and returns a "404 Not Found" HTTP code. Running the command yields:

```
[stdin]:
x [404] https://google.com/jobs.html | Failed: Network error: Not Found

1 Total, 0 OK, 1 Error (HTTP:1)
```

With our contribution two new flags are added to lychee: `--suggest` and `--archive`. The `--suggest` flag tells lychee to suggest link replacements for broken links, using a Web archive. The `--archive` flag specifies the Web archive to use for searching the link suggestions. Its default value, and currently only option, is "wayback" to use the Wayback Machine. The flag was added to allow easy extension in the future.

With the new Wayback Machine integration it is now possible to run the following command.

```
echo 'https://google.com/jobs.html' | lychee - --suggest
```

The output includes "suggestions" for each broken link recorded by the Wayback Machine in the past. If a link cannot be found by the Wayback Machine it will not be displayed.

```
[stdin]:
 x [404] https://google.com/jobs.html | Failed: Network error: Not Found

 Suggestions
https://google.com/jobs.html http://web.archive.org/web/20070623165349/http://
    www.google.com/jobs.html

1 Total, 0 OK, 1 Error (HTTP:1)
```

The suggestions are included in all the different output formats lychee knows: compact, detailed, json and markdown. To cover the new functionality, unit and integration tests were created. The Wayback Machine integration was merged on March 28th: Wayback Machine integration.

Concurrent archives

In the initial implementation of the Wayback Machine integration, the HTTP requests to the Wayback Machine were made in a sequential manner. This meant that every single request was awaited, before the next request was sent to the Wayback Machine. To reduce the time it takes to check all broken links for possible alternatives, we modified the previous implementation. We made use of the `for_each_concurrent`[22] function of the futures crate[23] in order to perform the HTTP requests in an asynchronous and concurrent way.

In addition, we added a CLI test (a kind of user acceptance test) for a previously uncovered code path. The changes were merged into the main branch on May 11th: Concurrent archives

Status codes in maps

lychee provides a `url` and a `status` property for each checked link in the result map. Previously, the `status` property was a string type. When running `lychee www.example.com --format json -v`, the result map contained the following `success_map`:

```
{
  "success_map": {
    "http://www.example.com/": [
      {
        "url": "https://www.iana.org/domains/example",
        "status": "OK (200 OK)"
      }
    ]
  }
}
```

However, for programmatic reuse of the result map, the numeric status code should be available. We changed the `status` property to an object with the properties `text` and `code`. The `text` property keeps the value of the original `status` property. The `code` property contains the numeric HTTP response status code. When running the above command again with our changes the result looks as follows.

[22] Run a stream to completion, executing the provided asynchronous closure for each element on the stream concurrently: https://docs.rs/futures/0.3.28/futures/stream/trait.StreamExt.html#method.for_each_concurrent

[23] A Rust implementation of futures and streams featuring zero allocations, composability, and iterator-like interfaces: https://docs.rs/futures/latest/futures/

```
{
  "success_map": {
    "http://www.example.com/": [
      {
        "url": "https://www.iana.org/domains/example",
        "status": {
          "text": "OK (200 OK)",
          "code": 200
        }
      }
    ]
  }
}
```

This enabled us to reuse lychee's output for the suggest command. The changes were merged on March 27th: Status codes in maps

6 Results

Testing of LMT with various sites and repositories has shown its link checking and suggestion features to be reliable. However, some sites use rate limiting or other mechanisms to restrict automated crawling of sites. In many cases, these URLs are classified as broken because the site-specific mechanisms are not directly handled by LMT. As a workaround, LMT can be configured to ignore the affected URLs. The detailed test results can be found in the appendix.

6.1 Release

We released LMT to the public on the 5th of June 2023. It is available on GitLab under the MIT licence[24]. We chose to release LMT under the MIT licence because it encourages open collaboration and allows flexible use. With the public release of LMT, we hope that individuals and organisations can benefit from the project's capabilities and functionality. Furthermore, we look forward to receiving feedback, suggestions, and contributions from the community to improve and extend LMT.

Our changes to lychee were released with version 0.12.0 and 0.13.0. By integrating the Wayback Machine into lychee, we have extended a well-known and established link checking tool, so that a wide range of users can benefit from it.

6.2 User stories

In the following it is described, if and how LMT implements the user stories listed in the Requirements chapter.

The first three user stories, tagged as "essential", are implemented with the `lmt check`, `lmt fix` and `lmt check --web` commands respectively. User story 4 is implemented with the metadata verification functionality. For user story 5 no complete solution is provided. However, an example of how to check for broken links with Continuous Integration and Continuous Deployment (CI/CD) pipelines is provided for GitLab and GitHub. These examples can be adapted so that user story 5 is fulfilled. Unfortunately, there was not enough time to implement user stories 6 and 8. User story 7 can partially be addressed by configuring LMT to ignore the "429 Too Many Requests" HTTP return status code, as explained in the Configuration file section.

lychee offers more possibilities to handle rate-limited sites. For example, concurrency can be reduced and a retry mechanism with an exponential backoff algorithm is used by default. [25] Additionally, lychee allows the configuration of basic authentication[26] with the use of the `--basic-auth` command line flag.[27]

This means that user stories 7 and 8 are addressed by lychee. Therefore, it is recommended to use lychee either in combination with LMT or standalone for user stories 7 and 8.

[24] A permissive free software licence: https://mit-license.org/
[25] How lychee handles rate-limiting: https://lychee.cli.rs/#/troubleshooting/rate-limits
[26] Basic HTTP authentication scheme RFC 7617: https://datatracker.ietf.org/doc/html/rfc7617
[27] lychee configuration options: https://lychee.cli.rs/#/usage/cli?id=options

6.3 Comparison with other tools

Broken link detection

Currently, LMT's link detection is quite conservative. Meaning that only URLs are detected which clearly are URLs. This means that "example.com" will not be detected as URL, because the scheme, such as "http" or "https" is not defined. In HTML and Markdown files, only hyperlinks formatted as such will be detected. For instance, in Markdown this includes the link syntax, while in HTML it corresponds to the "href" attribute. This behaviour is intended to prevent false positives, at the expense of missing potential URLs within text segments.

Checking websites In the following example LMT is compared with lychee and deadlinkchecker.com[28] when checking the website https://wikipedia.org for broken links. The results are displayed in table 5

Table 5: Check results of websites

Tool name	Total link count	Broken link count	Time
LMT	334	1	6s
lychee	335	4	5s
deadlinkchecker.com	652	3	11s

All the tools successfully identified the single broken link that was confirmed to be non-functional. deadlinkchecker.com performs additional checks on websites, including validating URLs within Cascading Style Sheets for correctness. It identified two broken links in the CSS file, which were confirmed to be broken. lychee erroneously reported three false positives by incorrectly identifying URLs as email addresses, as shown below.

```
1 Failed: Unreachable mail address: portal/wikipedia.org/assets/img/Wikipedia-logo
     -v2@1.5x.png: Invalid: Email does not exist or is syntactically incorrect
2 Failed: Unreachable mail address: portal/wikipedia.org/assets/img/Wikipedia-logo
     -v2@2x.png: Invalid: Email does not exist or is syntactically incorrect
3 Failed: Unreachable mail address: portal/wikipedia.org/assets/img/Wikinews-
     logo_sister@2x.png: Invalid: Email does not exist or is syntactically
     incorrect
```

This bug, causing false positives, was reported on GitHub: URL misinterpreted as email.

Checking local files Table 6 shows the results of checking the API Design Practice Repository[29] with LMT and lychee. The following pattern was used to check all Markdown and BibLaTeX files: `*/**/*.md *.bib`.

[28] Free broken link checking tool: https://www.deadlinkchecker.com/
[29] API Design Practice Repository: https://github.com/socadk/design-practice-repository

Table 6: Check results of local files

Tool name	Total link count	Broken link count	Time
LMT	629	21	79s
lychee	917	34	48s

Lychee is capable of detecting links within plain text in Markdown, without the Markdown syntax for URLs, which explains why it identified and checked a higher number of URLs compared to LMT. There are three reasons for the difference in the number of broken links detected. One link, namely in the "activities/SDPR-StepwiseServiceDesign.md" file, was not checked because it was not using the link syntax for Markdown. Four additional broken links were checked by lychee but not by LMT, because they are located inside of HTML comments. LMT currently does not check links within comments. lychee identified additional broken links that were not detected by LMT, specifically links to the website "https://doi.org". These links were flagged as broken, accompanied by the message "Failed: Too many redirects" in the checking result. Upon manual verification, these links were found to be fully functional, indicating that lychee's result regarding this site was inaccurate.

Detection of content drift

The metadata verification functionality of LMT allows the detection of content drift. Currently, the implementation tries to detect content drift of web pages by the document title. With this feature in LMT, any modifications to a website's title will result in an error, increasing user's confidence in the accuracy of their links. However, not all instances of content drift may be detected, as the website's body is not included as metadata for verification. Unfortunately, given the timeframe, we were not able to test the detection accuracy of the current implementation in its entirety. We have not identified any other tools with a content drift detection feature, making LMT unique in this regard.

Fixing broken links

LMT's "fix" command allows to quickly find and fix broken links for given files, provided that the Wayback Machine has archived the broken links in the past. This feature is unique, as we have found no other tools that have the ability to automatically replace broken links with working links to the Wayback Machine.

7 Discussion and summary

This chapter discusses the findings and shortcomings identified in the previous chapter and throughout the project.

7.1 Wayback Machine API stability

No issues were encountered regarding the reliability of the Wayback Machine during the development of LMT. However, after the extension of lychee, namely after the Concurrent archives feature was merged into the master branch of lychee, it was discovered that a test making use of the Wayback Machine API[30] was failing in some rare cases. Based on the investigation, the inconsistency observed during testing can be attributed to issues with the reliability of the Wayback Machine API. Since the test failure occurred so infrequently, no further modifications were made to the code or test.

The discussion and findings can be viewed on GitHub.

7.2 Support additional Internet archives

The aim of this project was specifically to integrate the Wayback Machine to handle broken links. Throughout the project, we found the Wayback Machine to be a highly capable Internet archive with a well implemented RESTful HTTP API. Our experience with the Wayback Machine was positive, as we encountered no shortcomings or drawbacks, with the exception of the rare stability problem mentioned above.

Nevertheless, there are other useful Internet archives which could be supported by both LMT and lychee in the future. Popular alternatives to the Wayback Machine are Archive.today[31] and the Memento Project[32].

The implementation of LMT and the extension of lychee have been designed with the addition of other Internet archives in mind. For lychee, we have already included a flag to specify which Internet archive should be used, although it is currently only possible to specify the Wayback Machine. The integration of additional Internet archives would benefit both projects.

7.3 Extension of LMT

Throughout the planning and implementation of LMT, we came across numerous ideas that have potential for further development.

Currently, HTML and Markdown files have designated link extractor classes in LMT so that links are reliably detected. If other file types are encounterd, a fallback class is used for link extraction. As discussed in the implementation chapter, this fallback implementation can be unreliable depending on the file type. Support for additional file types, such as reStructuredText, would make LMT more useful.

[30] Wayback Machine API: https://archive.org/help/wayback_api.php
[31] A time capsule for web pages: https://archive.is/
[32] An archive supported by the United States National Digital Information Infrastructure and Preservation Program: http://www.mementoweb.org/

Later, we implemented a metadata verification feature to detect content drift. For the implementation, we chose the website title as an indicator of content drift because it seemed like a simple but effective approach. Due to time constraints, we were unable to assess the detection accuracy of this approach. For future work, it would be useful to implement additional detection methods. These could for example include the content of web pages and make use of deep-learning algorithms. The accuracy of the different approaches could then be compared, so that content drift would be detected in the most reliable way.

During the implementation of the configuration file feature in LMT, we considered the inclusion of additional options. Several different ideas came up even before the feature was implemented In the end, we were able to implement the two options that we considered to be the most important. The following configuration options would be very useful additions to LMT:

- Connection timeout
- HTTP basic authentication[33] credentials
- Custom HTTP headers[34]

Finally, when checking websites, link detection could be improved by executing JavaScript to more closely mimic a web browser. At the moment, both LMT and lychee extract links directly from the HTML, without the execution of client-side JavaScript. This is a limiting factor as many modern websites are interactive single page applications that rely on client-side rendering. By adding the execution of JavaScript and emulating a web browser, for example with V8[35], it would be possible to reliably check single page applications and therefore provide a better link checking experience.

7.4 Plugin system for lychee

After discussions with the maintainer of the lychee project, it was decided that certain features, like the detection of content drift, do not align with the concept of lychee, despite their potential value. As a solution, the creation of a plugin system for lychee was proposed, so that lychee's functionality could be extended using separate plugins.

One way to implement such a system in lychee is to use the Extism[36] plugin system. Extism allows the creation of plugins with seven different programming languages. Before adding Extism to lychee, the plugin system should be specified in more detail. This would include the definition of types used for the communication between lychee and the plugins and the specification of hooks. Hooks are the different stages at which lychee could invoke the plugins, such as before checking a link and after checking a link. This would for example allow the creation of a plugin for lychee, capable of detecting content drift.

7.5 Summary

We developed LMT to provide an automated approach to detecting and fixing broken links and drifted content. Additionally, we extended the free and open source link-checking tool lychee. Early feedback from users and testing has shown that LMT is a useful tool. There are still areas for improvement and many ideas have been found during the project as to how LMT and lychee could be developed further.

[33] Basic HTTP authentication scheme RFC 7617: https://datatracker.ietf.org/doc/html/rfc7617
[34] Headers allow to pass additional information with HTTP requests: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers
[35] High-performance JavaScript and WebAssembly engine: https://v8.dev/
[36] A universal plug-in system: https://extism.org/

LMT has been released to the public on the 5th of June 2023 under the MIT licence. With the public release, we hope that individuals and organisations can benefit from the project's capabilities and functionality. Our additions to lychee have been released with versions 0.12.0 and 0.13.0, bringing the benefits of the integration with the Wayback Machine to a wider user base. Given the many ideas and potential improvements discussed in this chapter for both LMT and lychee, we look forward to future work that builds on LMT or explores the concepts presented in this chapter.

Glossary

Response status code

> The status code of a response is a three-digit integer code that describes the result of the request and the semantics of the response, including whether the request was successful and what content is enclosed (if any). All valid status codes are within the range of 100 to 599, inclusive. The first digit of the status code defines the class of response. The last two digits do not have any categorization role. There are five values for the first digit:
>
> - 1xx (Informational): The request was received, continuing process
> - 2xx (Successful): The request was successfully received, understood, and accepted
> - 3xx (Redirection): Further action needs to be taken in order to complete the request
> - 4xx (Client Error): The request contains bad syntax or cannot be fulfilled
> - 5xx (Server Error): The server failed to fulfill an apparently valid request

(Fielding, Nottingham, and Reschke 2022)

Sed

> Sed is a stream editor. A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline). While in some ways similar to an editor which permits scripted edits (such as ed), sed works by making only one pass over the input(s), and is consequently more efficient. But it is sed's ability to filter text in a pipeline which particularly distinguishes it from other types of editors.

(Fenlason et al. 2022)

Sed can perform many different commands on a stream of text. Using the `s/regexp/replacement/` syntax allows to replace text matching regular expressions. For example running `echo "Hello there" | sed "s/there/world/"` will result in `Hello world`.

Glob patterns

Globs, also known as glob patterns are patterns that can expand a wildcard pattern into a list of paths that match the given pattern.

> A string is a wildcard pattern if it contains one of the characters '?', '*' or '['. *Globbing is the operation that expands a wildcard pattern into the list of pathnames matching the pattern. Matching is defined by: - A '?' (not between brackets) matches any single character. - A '*'* (not between brackets) matches any string, including the empty string.

(Brouwer 2020)

As an illustration, consider the glob pattern `?at.*`. The pattern first matches a single character (case-insensitive) followed by the characters `at`. `*` matches any number of characters. So this pattern would match the following files: `Cat.png`, `Bat.jpg` and `cat.bmp`.

Exit codes

An exit code, or exit state, is an integer between 0 and 255 which is returned to the parent process by an executable program to indicate whether it was successfully executed. There are conventions for what sorts of status values certain programs should return. The most common convention is simply 0 for success and 1 for failure. Numbers from 2-255 can be used to represent various other negative results or problems.

(Free Software Foundation, n.d.)

C4

The C4 model, developed by Simon Brown, provides a practical and scalable approach to visualising and communicating the architecture of a software system. It provides a clear and concise way of understanding the structure, behaviour and interactions of the system at four different levels of abstraction.

The model consists of a series of hierarchical diagrams, including System Context, Container, Component and Class. Each diagram is based on the previous one, and provides an increasing level of detail and granularity.

(Brown 2018)

Bibliography

Agata, Teru, Yosuke Miyata, Emi Ishita, Atsushi Ikeuchi, and Shuichi Ueda. 2014. “Life Span of Web Pages: A Survey of 10 Million Pages Collected in 2001.” In *IEEE/ACM Joint Conference on Digital Libraries*, 463–64. https://doi.org/10.1109/JCDL.2014.6970226.

Brouwer, Andries. 2020. “Glob(7) — Linux Manual Page.” August 2020. https://man7.org/linux/man-pages/man7/glob.7.html.

Brown, Simon. 2018. “C4 Architecture Model.” June 2018. https://www.infoq.com/articles/C4-architecture-model/.

Fenlason, Jay, Tom Lord, Ken Pizzini, Paolo Bonzini, Jim Meyering, and Assaf Gordon. 2022. “Sed(1) — Linux Manual Page.” November 2022. https://man7.org/linux/man-pages/man1/sed.1.html.

Fielding, Roy T., Mark Nottingham, and Julian Reschke. 2022. “RFC 9110.” *RFC 9110: HTTP Semantics*. https://www.rfc-editor.org/rfc/rfc9110.html#name-status-codes.

Free Software Foundation, Inc. n.d. “Exit Status (the GNU c Library): Exit Status.” https://www.gnu.org/software/libc/manual/html_node/Exit-Status.html.

Goel, Vinay. 2016. “Defining Web Pages, Web Sites and Web Captures,” October.https://blog.archive.org/2016/10/23/defining-web-pages-web-sites-and-web-captures/ .

Lessig, Lawrence, Jonathan Zittrain, and Kendra Albert. 2014. “Scoping and Addressing the Problem of Link and Reference Rot in Legal Citations: How to Make Legal Scholarship More Permanent.” March 2014. https://perma.cc/D29D-MV4L.

World Wide Web Consortium. 2009. Web Addresses in HTML 5. May 2009. https://www.w3.org/html/wg/href/draft#url.

Zittrain, Jonathan, John Bowers, and Clare Stanton. 2021. “The Paper of Record Meets an Ephemeral Web: An Examination of Linkrot and Content Drift Within the New York Times.” April 2021. https://nrs.harvard.edu/URN-3:HUL.INSTREPOS:37367405.

Appendix

Test results

At the beginning of the project, the team has examined a list of example repositories and websites, which would be suitable for testing with LMT once it is in a stable state.

Most of the examples are in a way related to the Eastern Swiss University of Applied Science (Ost).

API Patterns Website

```
lmt check -w https://api-patterns.org/
```

```
[https://api-patterns.org/]:
https://twitter.com/leanpub/status/1383156883654021120/
https://twitter.com/api_patterns

101 total, 99 OK, 2 Errors
```

Conclusion

Checking a productive website with LMT works, however, since LMT isn't capable of recursively check further subpages, it would still be a big manual effort to check the whole site. Most of the link checking tools, which have a productive website checking support don't have such a feature either.

The result shows that something is not correct with two Twitter links. Twitter seems to implement a special mechanism, where it redirects requests, as long as a cookie is not set.

In this case it would make sense to ignore the two URLs by creating a configuration file.

Interface Refactoring

```
lmt check -w https://interface-refactoring.github.io/
```

```
[https://interface-refactoring.github.io/]:
https://www.ifs.hsr.ch/Architectural-Refactoring-for.12044.0.html?&L=4
https://au.linkedin.com/in/andreifurda
https://twitter.com/m_st

80 total, 77 OK, 3 Errors
```

Result

The result shows an actual broken link to https://www.ifs.hsr.ch. In this case, the website's certificate is not valid anymore.

The Twitter link is once again considered broken, because of their redirect mechanism. LinkedIn seems to implement a similar mechanism to prevent automatic website crawling as Twitter, but in the case of LinkedIn, they return an erroneous status code (999).

Suggest

Using suggest on the results of this check, returned following working suggestions:

```
https://www.ifs.hsr.ch/Architectural-Refactoring-for.12044.0.html?&L=4 http://
    web.archive.org/web/20220130062633/https://ifs.hsr.ch/Architectural-
    Refactoring-for.12044.0.html
https://twitter.com/m_st http://web.archive.org/web/20221226070737/https://
    twitter.com/m_st
```

Conclusion

LMT found a broken link and was able to suggest a working replacement.

Design Practice Repository

This repository is publicly available and contains .md source files, which are used to generate the website. The repository can therefore be checked locally using LMT, while limiting it to the Markdown files.

```
lmt check ./**/*.md
```

```
[./activities/DPR-ArchitecturalDecisionCapturing.md]:
http://resources.sei.cmu.edu/asset_files/Presentation/2012_017_001_31349.pdf

[./activities/DPR-ArchitectureModeling.md]:
http://agilemodeling.com/essays/barelyGoodEnough.html

[./activities/DPR-SMART-NFR-Elicitation.md]:
https://www.researchgate.net/publication/329760910
    _Capturing_Architectural_Requirements

[./artifact-templates/DPR-CRCCard.md]:
https://www.ifs.hsr.ch/Olaf-Zimmermann.11623.0.html?&L=4

[./background-information/README.md]:
https://www.researchgate.net/publication/220349352
    _Situational_Method_Engineering_State-of-the-Art_Review/link/0912
    f508a5a083e5bc000000/download
http://semat.org/what-is-it-and-why-should-you-care-
http://semat.org/essence-1.2
http://www.software-engineering-essentialized.com/home
https://download.eclipse.org/technology/epf/OpenUP/published/openup_published_1
    .5.1.5_20121212/openup/index.htm
https://ifs.hsr.ch/index.php?id=13195&L=4

and more...

614 total, 594 OK, 20 Errors
```

Conclusion

LMT displays broken links for every single file. The result again includes a couple of LinkedIn and Twitter links, which we know are false positives and could be ignored using a configuration file The Researchgate website uses similar redirect mechanism to prevent simple bots from scraping, as Twitter does. In this case it would also make sense to ignore the URLs by creating a configuration file.

Medium blog

```
lmt check -w https://medium.com/@docsoc
```

```
[https://medium.com/@docsoc]:
https://medium.com/followers?source=user_profile
    ------------------------------------
https://medium.com/how-to-create-architectural-decision-records-adrs-and-how-not
    -to-93b5b4b33080?source=user_profile---------0----------------------------
https://medium.com/how-to-create-architectural-decision-records-adrs-and-how-not
    -to-93b5b4b33080?source=user_profile---------0----------------------------
https://medium.com/api-patterns-website-redesigned-and-sample-book-chapter-
    available-df9daf4b5e15?source=user_profile
    ---------1----------------------------
https://medium.com/api-patterns-website-redesigned-and-sample-book-chapter-
    available-df9daf4b5e15?source=user_profile
    ---------1----------------------------
https://medium.com/a-checklist-for-api-design-reviews-5f7db45b0cb3?source=
    user_profile---------2----------------------------
https://medium.com/a-checklist-for-api-design-reviews-5f7db45b0cb3?source=
    user_profile---------2----------------------------

and many more...

130 total,  77 OK,  53 Errors
```

Result

According to LMT, a lot of links were broken here. Upon manual inspection, it was found that the links were actually functional. After having a closer look, at the problem here, it was figured out, that medium automatically redirected “https://medium.com/@docsoc” to “https://docsoc.medium.com/” and thus, the root path which is reused for the relative links on the website was wrong, and therefore couldn’t resolve the links.

Retrying with correct URL

```
lmt check -w https://docsoc.medium.com/
```

```
130 total, 130 OK, 0 Errors
```

Conclusion

When a website containing relative links redirects itself, the wrong root path is used by LMT to resolve the relative links. This bug was not fixed until the end of this thesis. As a workaround, the resolved URL should be used instead of the redirecting URL.

Cloud Application Lab

```
lmt check -w https://www.ost.ch/de/forschung-und-dienstleistungen/informatik/ifs-institut-fuer-software/labs/cloud-application-lab
```

Result

The server seems to implement a kind of rate-limiting, checking this website resulted in a temporary ban on https://www.ost.ch related pages. This is a technical limitation of the process of checking productive websites which could only be bypassed by rate-limiting LMT itself.

Conclusion

The results in this case are inaccurate. The OST website could be ignored in the config file, so that this won't happen again.

ozimmer.ch

```
lmt check -w https://ozimmer.ch
```

```
67 total, 67 OK, 0 Errors
```

Conclusion

No broken links detected

Jolie Language Documentation

```
lmt check -w https://www.jolie-lang.org/
```

```
[https://www.jolie-lang.org/]:
https://twitter.com/jolielang
https://www.jolie-lang.org/news
https://pixis.co/
mailto:webmaster@jolie-lang.org

40 total, 36 OK, 4 Errors
```

Result

The result once again includes a Twitter link, which can be ignored using a configuration file. The other links are broken.

Suggest

```
lmt check -w https://www.jolie-lang.org/ | lmt suggest
```

```
https://twitter.com/jolielang http://web.archive.org/web/20190930232612/https://
    twitter.com/jolielang
https://www.jolie-lang.org/news http://web.archive.org/web/20220129051520/https
    ://www.jolie-lang.org/news
https://pixis.co/ http://web.archive.org/web/20220625170906/https://pixis.co/
```

Conclusion

LMT found broken links on the productive website, and was able to suggest working replacements. The Twitter link is once again considered broken, because of their redirect mechanism.

Project plan

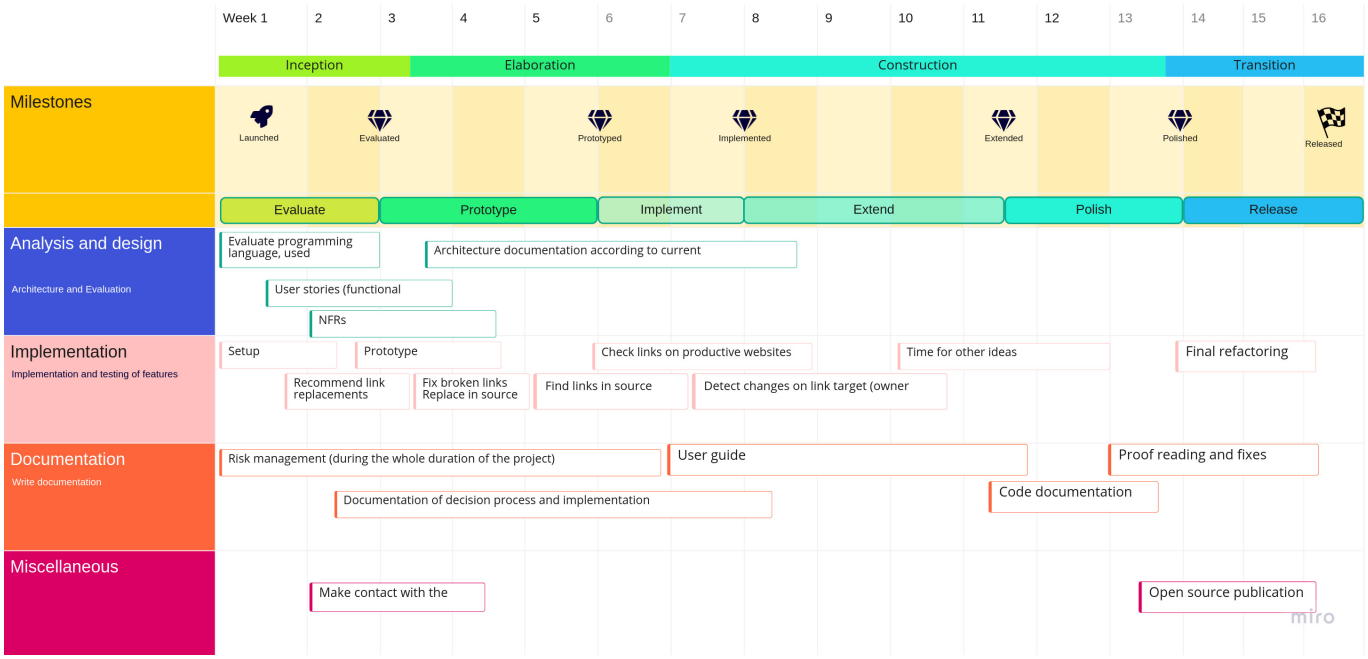

Figure 10: Project plan

As shown in figure 10 phases and milestones were defined to help keep the project on track. Each phase ends with a milestone to mark the end of the phase. Four different layers were defined to give a better overview of the different project tasks.

Milestones

Below is the definition of each milestone. The definition specifies which tasks must be completed in order for the milestone to be considered completed.

Evaluated

- Tech stack was defined
- Documentation repository was set up
- Core repository was set up

Prototyped

- Prototype with previously defined tech stack exists
- Most important user stories & NFRs were defined
- Third-party integrations were tested

Implemented

- Most essential user stories covered (potentially with the use of a third-party program)
- Most essential features implemented

Extended

- Additional user stories implemented
- Additional features implemented

Polished

- User guide finished
- Code ready to be released
- Code documented where necessary

Released

- Licence for the project is defined
- Project published as FOSS project, publicly available on GitLab or GitHub

Risk management

Table 7 is used to document technical and project risks. The risk's likelihood of occurrence and its impact is estimated and ranked using a numerical score between 1 (Lowest) and 5 (highest). The risk's severity score is based on the likelihood and impact rating. For each risk, a mitigation and contingency action is defined.

Table 7: List of possible risks

#	Description	Likelihood	Impact	Severity	Action
1	Wayback Machine API[37] availability issues	2	5	4	**Mitigation**: Assess the reliability of the Wayback Machie API and identify alternative web archives. **Contingent**: Use an alternative source of data.
2	Wayback Machine API changes	1	3	2	**Mitigation**: Create automated tests and sign up to mailing list. **Contingent**: Adapt code to API changes.
3	Target group and assumptions about user behaviour change during the project, so that the requirements and tech choices are no longer adequate	3	3	3	**Mitigation**: Creation of a prototype. **Contingent**: Change tech stack.
4	Requirements turn out to be more complex than planned	2	3	3	**Mitigation**: Careful planning and react quickly when encountering problems. **Contingent**: Reduce the scope of the project.
5	lychee[38] has less functionality than planned	1	4	3	**Mitigation**: Test lychee's functionality before committing to contribute and use lychee. **Contingent**: Fully focus on the prototype, change the scope of the project accordingly.
6	lychee cannot be extended, because the team is not very experienced with the employed technologies and libraries and the overall complexity was underestimated	2	3	3	**Mitigation**: Assess lychee's code and architecture before committing and contact community or main developer when encountering problems. **Contingent**: Look for a different link checker to extend or focus fully on the prototype instead.
7	Changes to lychee are rejected or ignored because the code is not accepted or the community or main developer is inactive	2	2	2	**Mitigation**: Check how active lychee's community is and contact the main developer before committing to contribute to lychee. Also create small pull requests early and improve them iteratively with the help of the community instead of creating big pull requests. **Contingent**: Focus on the prototype instead of lychee or work with the forked project rather than the upstream version of lychee.

[37] Wayback Machine API: https://archive.org/help/wayback_api.php
[38] Open source broken link checking tool: https://github.com/LycheeOrg/Lychee/

LMT user guide

Installation

Pre-built binaries (recommended) We provide binaries for Linux, macOS, and Windows for every release. The binaries can be downloaded from the releases page.

Make accessible from any location The rest of this guide assumes that LMT has been added to the PATH. If you skip this part you will have to execute LMT with specifying its location. So instead of just invoking `lmt` you will need to run `./lmt` when the binary is located in the current working directory.

Adding an executable to the PATH allows for easy and direct execution of the command from any directory in the command-line interface without specifying its full path.

- Rename the downloaded binary to `lmt`(`lmt.exe` on Windows)
- Follow this guide to add the directory to PATH: Add directory to PATH

Compile LMT yourself Firstly, make sure that Deno is installed locally:

Deno installation

And this repo has to be cloned on your site as well:

```
git clone git@gitlab.com:lmt7360522/lmt.git
```

Finally, run the build command:

```
deno task compile
```

The binary should now be created in the project's source. To check whether it works, run the following command from the project's source directory:

```
./lmt --help
```

To make LMT accessible from any location, follow the steps in Make accessible from any location

Usage

Make sure you have installed LMT, as described in Installation Currently, it's only possible to use LMT from the command line.

Check for broken links The main use case of LMT is to check for broken links in source files.

To check a local file for broken links, use:

`lmt check <path-to-file>`, for example `lmt check README.md`

Glob patterns are also supported by LMT, which allows the checking of multiple files at once:

`lmt check <glob-pattern>`, for example `lmt check foldername/**.html`

It is important to note that when utilizing the pattern from the same directory where the file(s) that should match the pattern are located, there is generally no need to include a `./`prefix. In the context of glob patterns, the `./` prefix holds the same significance as it does in regular expressions, potentially matching a file with precisely one character.

Instead of checking local files, it is also possible to check links on websites:

`lmt check -w <url>`, e.g. `lmt check -w https://www.example.com`

Suggest changes LMT is capable of suggesting replacements for broken links, which exist on the WayBack machine.

The suggest command can be used by:

- piping the output of a check command:

After successfully checking a file for broken links, the output can be piped into the `suggest` command. LMT will now search for replacements in the WayBack machine for each of the broken links provided. `lmt check <glob-pattern> | lmt suggest`, e.g. `lmt check foldername/**.html | lmt suggest`

- providing the links as arguments:

`lmt suggest <broken-url(s)>`, e.g. `lmt suggest "https://www.example.com https://www.ost.ch"`
Note that when entering multiple urls manually, they have to be newline separated (Each link on a single line). In most of the terminals, this can be achieved by wrapping the urls in quotation marks (" ").

Fixing files automatically Results of the `suggest` command provide recommendations for changes of broken links, which can be automatically replaced in all files provided, when using `lmt fix`

`lmt fix <glob-pattern>`, e.g. `lmt fix src/**/*.md`

Providing a link map If you want to check source files with another link checker, or have a file prepared with mappings of "broken links" to "replacement links", you can simply use this map as an argument for `lmt fix`

`lmt fix --link-map "<broken-link> <replacement-link>"`, e.g. `lmt fix --link-map "https://www.example.com/broken https://www.example.com/working"`

Using with lychee The output of `lychee --suggest` can be used as the link-map argument for `lmt fix`. In that case, LMT will only replace the links, which were checked and suggested by lychee.

`lmt fix --link-map "$(lychee <file(s)> --suggest)"<file(s)-to-replace>`

Help The help command can be used after each command, and will display the possible arguments which can be applied to any command.

Usage

`lmt -h`

```
lmt check -h
lmt suggest -h
lmt fix -h
```

Configuration file

LMT recognises `.lmt.json` as config file if it is located in the same directory where LMT is run. The path is currently not configurable, but feature requests and contributions are welcome. The config file must be valid JSON. The root object must contain the `"ignore"` and `"ignoreMetadataForUrls"` property.

With the `"ignore"` option it's possible to specify status codes which should be ignored for a given URL. It's also possible to ignore all status codes with the option `"all"`. The value of `"ignore"` is an object, where all keys specify a URL as `string` and all values are of type `StatusCodes`. `StatusCodes` is either an array of status codes or `"all"`. (`type StatusCodes = Array<number> | 'all'`)

The `"ignoreMetadataForUrls"` option specifies an array of URLs which should be ignored when using the check command with the `--create-snapshot` or `--verify-snapshot` flag.

In the following example, three URLs are configured to be ignored by LMT. For the last URL all results should be ignored, (including DNS errors, connection errors, etc.) so that effectively the whole URL is ignored. Additionally, the metadata snapshot functionality should be disabled for one URL.

```
{
  "ignore": {
    "https://example.com": [
      418,
      500
    ],
    "https://example.com/something": [
      404
    ],
    "https://deno.land/std@0.182.0/": "all"
  },
  "ignoreMetadataForUrls": [
    "https://example.com"
  ]
}
```

CI/CD

LMT is designed to integrate seamlessly with Continuous Integration and Continuous Deployment (CI/CD) pipelines. Using the "check" command for example, ensures that broken links are caught early in the pipeline, preventing them from reaching production environments. Any command of LMT can be easily integrated into different stages of a pipeline.

Docker image We have published a Docker image containing the binary executable of the Link Management Tool on DockerHub, which allows LMT to be easily integrated into CI/CD pipelines.

General script When using the Docker image, any LMT command can be run. For example `lmt check <filename(s)>` can be used to check for broken links.

Failures If a use case requires the pipeline to use or return the correct exit code, `set +e` should be added at the beginning of the script, so that it will not exit immediately, if a non-zero exit code was returned by a command. This could be the case, if a script should behave differently, when broken links have been detected or when a pipeline is allowed to fail on occurrence of specific exit codes.

GitLab

.gitlab-ci.yml:

```
stages:
  - check-for-broken-links

check-for-broken-links-job:
  image:
    name: thomaszahner/lmt
    entrypoint: [ "" ]
  stage: check-for-broken-links
  script:
    - lmt check ./**/*.md
```

GitHub

An example GitHub Action was created to test and demonstrate the functioning of LMT inside a GitHub Action. Note that it is expected the pipeline to fail, since a couple of broken links have been added to the demo files.

ci.yml:

```
name: LMT

on:
  push:
    branches:
      - master
      - main
  pull_request:
    branches:
      - master
jobs:
  check-for-broken-links:
    runs-on: ubuntu-latest
    container:
      image: thomaszahner/lmt
    steps:
      - uses: actions/checkout@v1
      - name: Check for broken links
        run: |
          lmt check ./**/*.md
```

Input and output format reference

Default

The default format is used as output for the check command, or as input for the suggest command. It is meant to be a human-readable format and consists of a space separated list of urls in the following form. `[<url>...]`;

Example:

```
https://www.example.ch/ https://www.ost.ch/
```

Usage: `lmt suggest https://www.example.ch/ https://www.ost.ch/`

JSON

The JSON format can be used as output for the check command, or as input for the suggest command. It consists of a JSON object, with at least a property called `fail_map` and `success_map`. They both are an object with key-value pairs of relative file paths as the key, and an array of URL objects as the value. The URL object contains at least an `url` string property, but can also include a `status` string property.

This format is meant to be used by the suggest command or third-party software.

Example:

```
{
  "success_map":{},
  "fail_map": {
    "./readme.md": [
      {
        "url": "https://www.example.com",
        "status": 404
      }
    ]
  }
}
```

The JSON input is adapted to lychees JSON output format and therefore implicitly compatible, which means that json output of a lychee command can directly be used with LMT.

Default suggest output

The default output format of a suggest command consists of the target URL, followed by a space, which is then followed by the suggested URL. A separate line is used for each target URL.

Example:

```
www.example.com https://web.archive.org/web/20230307000355/http://www.example.
   com/
www.ost.ch https://web.archive.org/web/20230202101609/https://www.ost.ch/
```

JSON suggest output

The suggest command's JSON output format consists of an array of objects, which each contains the properties `url` and `suggestion`

Example:

```
[
  {
    "url": "www.example.com",
    "suggestion": "https://web.archive.org/web/20230307000355/http://www.example.com/"
  },
  {
    "url": "www.ost.ch",
    "suggestion": "https://web.archive.org/web/20230202101609/https://www.ost.ch/"
  }
]
```

CSV suggest output

A comma-separated string, which is compatible with common spreadsheet programs.

Example:

```
"Url","Suggestion"
"www.example.com","https://web.archive.org/web/20230307000355/http://www.example.com/"
"www.ost.ch","https://web.archive.org/web/20230202101609/https://www.ost.ch/"
```

Sed suggest output

The "sed" output format is designed to be used in combination with the sed command. When doing so, the following flags must be used with sed:

- `--in-place` (`-i`) which specifies to directly replace matched content in-place
- `--extended-regex` (`-E`) which specifies that the expression is written in the POSIX-Extended Regular Expression (ERE) flavor

Example:

```
s/([^][!*'();:@&=+$,/?%#[:alnum:]]+|[[:space:]]+|^)(www.example.com)([^][!*'();:@&=+$,/?%#[:alnum:]]+|[[:space:]]+|$)/\1https://web.archive.org/web/20230307000355/http://www.example.com/\3/g;s/([^][!*'();:@&=+$,/?%#[:alnum:]]+|[[:space:]]+|^)(www.ost.ch)([^][!*'();:@&=+$,/?%#[:alnum:]]+|[[:space:]]+|$)/\1https://web.archive.org/web/20230202101609/https://www.ost.ch/\3/g
```

Usage:

```
sed -i -E "$(cat testLinks | lmt suggest --output-format sed)" example.html
```